

ALEODOR - DANIEL IOAN

**BAZELE FUNCȚIONĂRII
ȘI PROGRAMĂRII
CALCULATOARELOR**

**Ediția a II-a
(revizuită și completată)**

**Editura POLITEHNIUM
Iași, 2021**

ALEODOR-DANIEL IOAN

**BAZELE FUNCȚIONĂRII
ȘI PROGRAMĂRII
CALCULATOARELOR**

EDIȚIA A II-A

(REVIZUITĂ ȘI COMPLETATĂ)

EDITURA POLITEHNIUM

IAȘI, 2021

Editura POLITEHNium

Universitatea Tehnică “Gheorghe Asachi” din Iași
Str. Prof. dr. doc. Dimitrie Mangeron, nr. 67, 700050 Iași, România
Tel: 40 232 – 212324
Fax: 40 232 – 211667
E-mail: cercetare@tuiasi.ro

Editura Politehnium (fostă “Gheorghe Asachi”) este recunoscută de
Consiliul Național al Cercetării Științifice din Învățământul Superior (CNCSIS)

Coordonator editură: Prof. univ. dr. ing. Carmen LOGHIN

Referenți științifici:

Prof. univ. dr. ing. **Alexandru ONEA**
Universitatea Tehnică *Gheorghe Asachi* din Iași
Facultatea de Automatică și Calculatoare

Conf. univ. dr. ing. **Mihai POSTOLACHE**
Universitatea Tehnică *Gheorghe Asachi* din Iași
Facultatea de Automatică și Calculatoare

Tehnoredactare și grafică:

Șef lucr. univ. dr. ing. fiz. Aleodor-Daniel IOAN

*Răspunderea pentru tot ceea ce conține prezenta carte
aparține în întregime autorului (autorilor) acesteia.*

Descrierea CIP a Bibliotecii Naționale a României

IOAN, ALEODOR DANIEL

Bazele funcționării și programării calculatoarelor / Aleodor-Daniel

Ioan. - Ed. a 2-a, reviz. și completată. - Iași : Politehnium, 2021

Conține bibliografie

ISBN 978-973-621-498-1

004

*Mamei mele Valentina și tatălui meu Ion,
care m-au susținut mereu pe căile cunoașterii*

Cuprins

Prefață la ediția a II-a	7
1. Introducere	11
2. Sisteme de numerație	13
2.1. Trecerea unui număr dintr-o bază în alta	15
2.1.1. Trecerea din baza B în baza zece (în zecimal)	15
2.1.2. Trecerea unui număr <i>întreg</i> din zecimal în baza B	17
2.1.3. Trecerea unui număr <i>subunitar</i> din zecimal în baza B	20
2.1.4. Trecerea unui număr cu virgulă fixă din zecimal într-o bază oarecare B	22
2.1.5. Trecerea unui număr din baza R în baza B când B este o putere a lui R	22
2.1.6. Trecerea unui număr din baza B în baza R când B este o putere a lui R	23
3. Reprezentarea informației în sistemele de calcul	25
3.1. Reprezentarea numerelor fără semn în virgulă fixă	25
3.2. Reprezentarea numerelor <i>întregi</i> cu semn	26
3.2.1. Reprezentarea în mărime și semn	27
3.2.2. Reprezentarea în cod complementar față de 1	28
3.2.3. Reprezentarea în cod complementar față de 2	31
3.3. Reprezentarea directă a cifrelor zecimale	37
3.4. Reprezentarea în virgulă mobilă a numerelor reale	39
3.5. Reprezentarea caracterelor alfa-numerice	43
4. Operații aritmetice binare	45
4.1. Operații cu numere fără semn (FS)	45
4.2. Operații între numere cu semn în C_2	47
4.2.1. Numere pozitive	48
4.2.2. Numere negative	49
4.2.3. Numere cu semne opuse	51

4.3. Operații cu numere reprezentate BCD	53
5. Algebra Booleană și algebra logică	57
5.1. Porți logice elementare	58
5.2. Porți logice ne-elementare	61
5.3. Teoremele algebrei logice	63
5.4. Operații logice pe mai mulți biți	65
5.5. Operații aritmetico – logice de rotație	66
6. Formele canonice ale funcțiilor logice	69
6.1. Forma canonică disjunctivă (<i>FCD</i>)	69
6.2. Forma canonică conjunctivă (<i>FCC</i>)	70
7. Minimizarea grafică a funcțiilor logice	73
7.1. Diagrame Veitch – Karnaugh	73
7.2. Funcții logice incomplet definite	75
8. Etajul de ieșire al porților logice; magistrale	77
9. Câteva funcții logice particulare	83
9.1. Decodificatoare (de adresă)	83
9.2. Multiplexoare	84
9.3. Demultiplexoare	86
9.4. Sumatoare	88
10. Structuri logice secvențiale	91
10.1. Structuri secvențiale cu memorie (celula RS)	91
11. Perfecționarea celulei RS	97
11.1. Precizarea momentului de comandă	97
11.1.1. Celula RS de tip "LATCH" (zăvor)	97
11.1.2. Bistabilul RS sincron pe front	98
11.2. Eliminarea nedeterminării; bistabilul <i>JK</i>	99
11.3. Comanda simultană a intrărilor; bistabilii <i>T</i> și <i>D</i>	101
11.4. Eliminarea nedeterminării și comanda simultană; "Latch"-ul de tip <i>D</i>	101
12. Aplicații directe ale bistabilelor	103
12.1. Numărătoare	103
12.1.1. Numărătoare asincrone	104

12.1.2. Numărătoare sincrone _____	105
12.2. Registre _____	106
12.2.1. Registre cu încărcare asincronă _____	106
12.2.2. Registre cu încărcare sincronă _____	107
12.2.3. Registre de deplasare / rotație _____	109
12.3. Numărătoare cu încărcare _____	110
13. Memoria sistemelor de calcul _____	111
13.1. Memorii de tip <i>ROM</i> _____	112
13.2. Memorii de tip <i>SRAM</i> _____	115
14. Organizarea unui sistem de calcul _____	119
15. Microprocesorul 8085 _____	123
15.1. Arhitectura internă _____	123
15.2. Funcționarea microprocesorului _____	128
16. Stiva hardware, subrutine și întreruperi _____	133
16.1. Păstrarea temporară a datelor _____	133
16.1.1. Introducerea datelor în stivă _____	133
16.1.2. Extragerea datelor din stivă _____	135
16.2. Stiva și subrutinele _____	137
16.3. Stiva și întreruperile _____	139
17. Întreruperile microprocesorului 8085 _____	141
18. Dispozitive IO și configurația sistemului _____	145
18.1. Dispozitive de intrare / ieșire _____	145
18.2. Utilizarea "latch"-ului de 8 biți cu 3 stări <i>HCT 573</i> _____	147
18.2.1. Utilizarea la memorarea adreselor inferioare _____	147
18.2.2. Utilizarea ca port de intrare _____	148
18.3. Configurația unui sistem minimal de calcul _____	150
18.3.1. Componentele și rolul lor _____	151
18.3.2. Logica de selecție și maparea memorie / intrare – ieșire _____	152
19. Probleme și subiecte teoretice propuse _____	155
Bibliografie selectivă pe capitole _____	181

Prefață la ediția a II-a

Cartea încearcă să sintetizeze într-un ansamblu autonom toate cunoștințele introductive și de nivel mediu necesare pentru *înțelegerea alcătuirii interne, a modului de funcționare și a principiilor care stau la baza programării sistemelor de calcul*. Ea poate fi utilizată de oricine dorește să înțeleagă cum se poate construi un sistem de calcul, să afle după ce principii funcționează și care este legătura dintre logica hardware și programarea software. Pentru studenții de la facultățile de **Automatică și Calculatoare**, lucrarea constituie un curs la început de studii, pentru care nu sunt necesare alte cunoștințe specializate, ci doar câteva noțiuni de matematică elementară și mai ales un mod de gândire logic și coerent. Ea poate fi folosită și de către studenții facultăților de *Electronică și Telecomunicații*, dar acolo aceste noțiuni sunt transmise mai detaliat, împărțite pe parcursul a mai multor discipline de studiu.

Tematica aleasă urmărește o expunere esențializată a vastei problematice din domeniu, plecând de la ideea că *pentru programarea eficientă a unui sistem de calcul este necesar să se cunoască cum funcționează, iar pentru a se putea înțelege funcționarea sa, trebuie să se știe mai întâi cum a fost construit*. Conținutul materialului a fost elaborat în vederea învățării prin construirea graduală a cunoștințelor, cu acumularea lor piramidală într-un edificiu coerent, care să aibă finalitate evidentă. Pentru detalierea alcătuirii sistemelor de calcul se merge pe o prezentare ierarhică, frecvent întâlnită ca tehnică de proiectare, în care modulele construite cu ajutorul unor componente elementare devin la rândul lor elemente de bază în module aflate pe nivelele ierarhice superioare.

Expunerea este adaptată la tendințele moderne de învățare după o prezentare schematică, predominant grafică, având numeroase desene explicative și mai puțin text propriu-zis sau formule. Pentru a ușura asimilarea de către studenți a noțiunilor nu tocmai simple, s-a depus un efort semnificativ pentru tehnoredactarea cât mai didactică a materialului, prin schematizarea cu săgeți explicative a desenelor și clasificărilor, sublinierea prin înconjurare și colorare a simbolurilor din formule și textelor corespunzătoare unor aspecte importante. O serie de noțiuni sunt explicate *într-un mod inedit și complet original*, cu avantaje evidente în privința înțelegerii funcționării, care nu a mai fost găsit de autor în alte lucrări similare. Cum problematica atinsă este bine cunoscută și a fost tratată în foarte multe

lucrări din domeniu, *originalitatea materialului în ansamblu* vine tocmai din felul de organizare a tematicii selectate, din modul de explicare diferit al aspectelor ingineresti fundamentale discutate, precum și din prezentarea schematică sub o formă grafică inedită.

Deși s-a încercat esențializarea tematicii și utilizarea unor explicații cât mai simple, nu toate componentele pot fi descărcate complet de detalii și aduse la nivelul de “cutie neagră”, adică de structură necunoscută având doar intrări și ieșiri, pentru că nu ar mai putea fi complet înțelese sau și-ar pierde aplicativitatea clară și capacitatea de a se constitui într-un tot unitar. În viziunea autorului, omiterea completă a detaliilor de alcătuire internă pentru unele blocuri componente importante poate conduce la introducerea bruscă a unor cunoștințe dispartate, în loc de înțelegerea graduală a legăturilor cu noțiunile anterioare, care nu ar fi cea mai bună soluție didactică.

Din aceste motive, conținutul poate părea puțin prea larg pentru un curs introductiv, dar trebuie avut în vedere că nu toată materia este predată efectiv în cadrul prelegerilor, insistându-se mai mult asupra aspectelor importante și detaliilor esențiale, în funcție de nivelul pregătirii și de reacția studenților. Materialul este însă complet, pentru a oferi tuturor celor interesați detalii suficiente pentru o eventuală documentare ulterioară sau formarea legăturilor cu celelalte discipline din planul de învățământ.

În ediția a II-a au fost aduse modificări consistente cu privire la mai multe aspecte:

- corectarea unor greșeli care s-au strecurat în prima ediție mai ales din cauza dificultății de tehnoredactare, în mare majoritate grafică;
- ajustarea exprimării și adăugarea unor desene sau explicații adiționale sub formă de text, acolo unde prezentarea era prea schematizată;
- introducerea anumitor noțiuni suplimentare și detalierea altora, cu scopul înțelegerii cât mai unitare și mai complete a sistemelor de calcul;
- un capitol complet nou cu probleme propuse pentru verificarea cunoștințelor, o parte fiind dintre cele date la examenele cu studenții, pe parcursul a mai multor sesiuni.

Față de prima ediție, s-a reușit explicarea pe scurt a *construcției și funcționării tranzistoarelor* utilizate în tehnica de calcul drept comutatoare comandate, într-un mod simplificat și mai mult la nivel intuitiv, fără a apela deloc la noțiuni de electronică digitală sau fizica semiconductorilor, dar suficient pentru a înțelege de la ce se pleacă în funcționare.

De asemenea, în ediția a II-a s-a introdus un sub-capitol despre *reprezentarea numerelor în virgulă mobilă*, care constituie baza tuturor aplicațiilor de grafică 3D, jocuri, multimedia și calcul științific pe dispozitivele digitale moderne. S-a încercat însă și aici păstrarea liniei generale a lucrării, cu anumite detalii și exemple, ca să se înțeleagă în ce

constă standardul internațional consacrat IEEE 754 și nu doar la nivelul unui principiu general de reprezentare. Schematizarea detaliilor esențiale și prezentarea lor la nivel didactic a constituit o provocare chiar și pentru experiența autorului, conducând și aici la o sinteză originală față de majoritatea lucrărilor, care tratează această reprezentare într-un mod prea sofisticat.

În scopul exemplificării funcționării unui microprocesor, s-a ales “bătrânul” **8085** pentru că este *microprocesorul real cu arhitectura internă cea mai simplă*, prezent încă în producție industrială și utilizat în scopuri didactice de către multe instituții de învățământ. Schema detaliată a unui sistem minimal de calcul cu un astfel de microprocesor a fost gândită inclusiv pentru a putea fi realizată practic de cei pasionați, care doresc să experimenteze suplimentar realizarea și programarea unui echipament propriu. Pe schema propusă, s-au evitat intenționat toate circuitele de intrare / ieșire dedicate care nu mai sunt actuale și au fost înlocuite prin circuite elementare de tipul celor prezentate în lucrare.

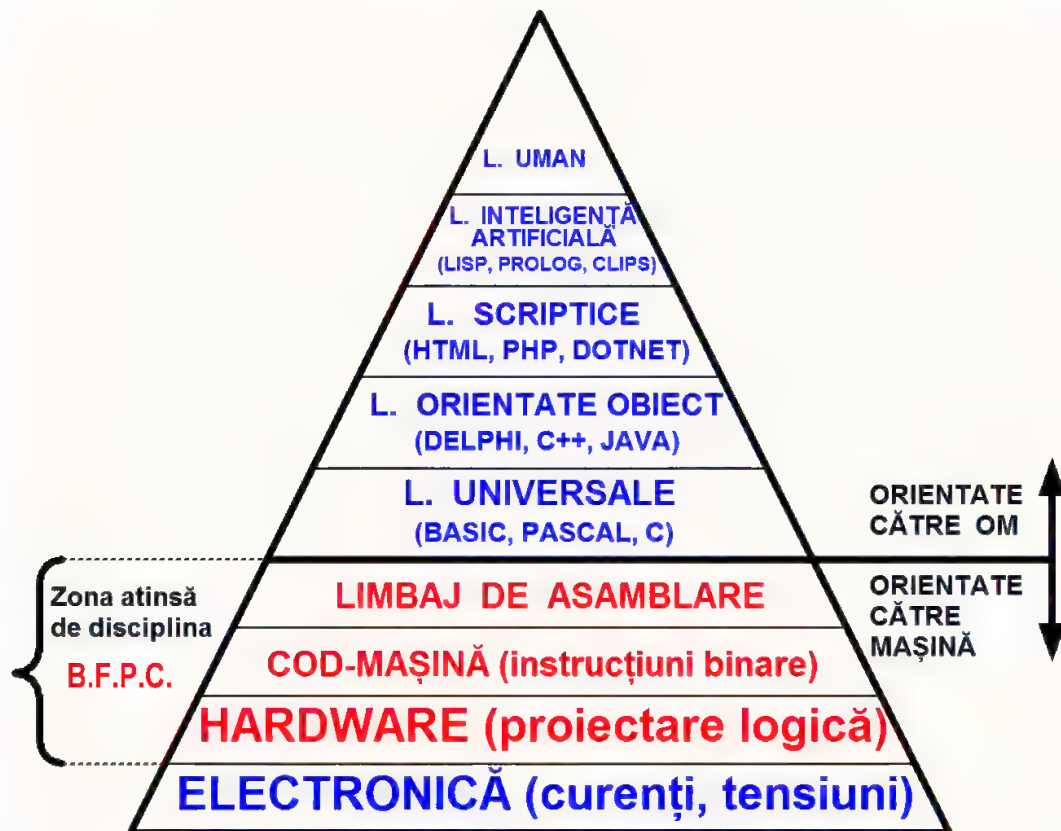
Aleodor Daniel Ioan ,

Iași, aprilie 2021 .

1. Introducere

Obiectivele disciplinei: – Învățarea unor noțiuni de bază cu privire la **alcătuirea, funcționarea și programarea elementară** a sistemelor de calcul.

PIRAMIDA SISTEMELOR DE CALCUL (SOFT-HARD):



Definiție: – **Informația** este expresia distribuției neuniforme a materiei și energiei în spațiu și timp, care caracterizează gradul de organizare al unui sistem finit.

– Gradul de incertitudine eliminat prin producerea unui eveniment se numește **cantitate de informație** și are ca unitate de măsură **bitul**:

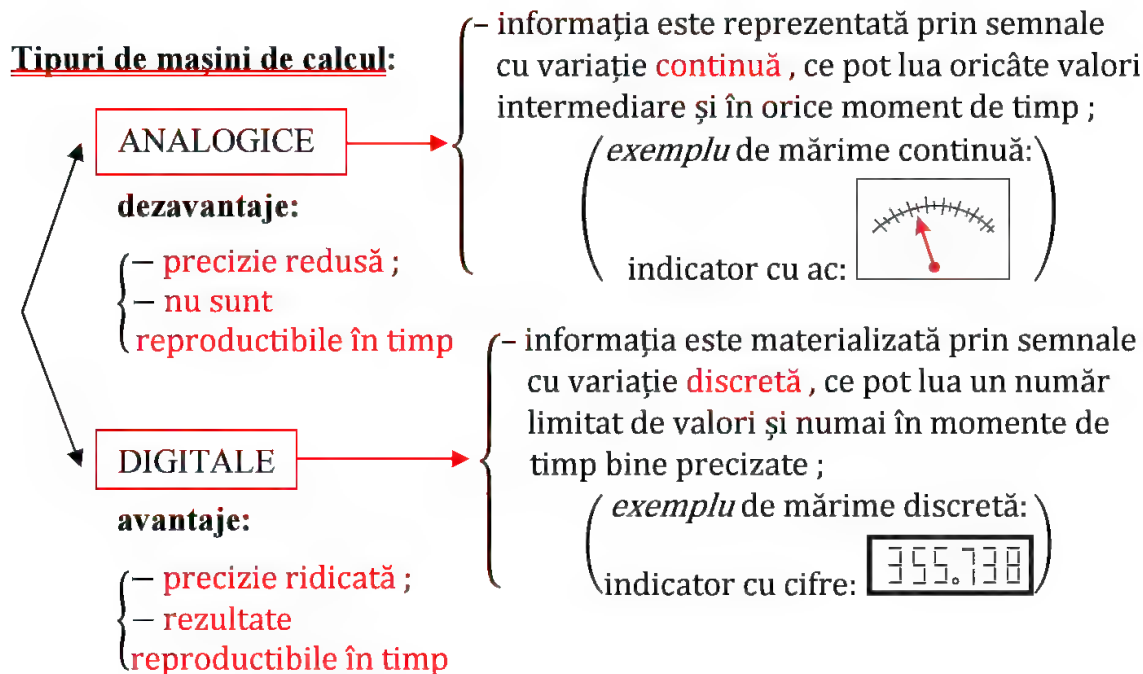
1 bit = cantitatea de informație transmisă la producerea unui eveniment dintre 2 evenimente posibile, cu aceeași probabilitate de apariție.

Exemplu: – Aruncarea unei monede cu 2 fețe:
 ↗ Fața cu *valoarea* (banul)
 ↘ Fața cu *simbolul* (marca)

- Înainte de producerea evenimentului existau 2 posibilități cu probabilitate egală de apariție ;
- După producerea evenimentului se cunoaște pe care față a căzut moneda $\Rightarrow$ se cunoaște o cantitate de informație cu valoarea de **1 bit**.

Elementele constitutive ale oricărui sistem de calcul: – Au fost enunțate prima dată de matematicianul **VON NEUMANN** (mașina “**NEUMANN**”):

1. Trebuie să existe un **mediu de intrare**, prin intermediul căruia să se introducă **instrucțiuni și date**;
2. Mașina trebuie să conțină o **unitate de memorie**, în care să se păstreze **programul** (ansamblul de instrucțiuni) și datele de lucru;
3. Este necesară existența unei **unități de control**, capabilă să **interpreteze** instrucțiunile-mașină și să comande celelalte elemente constitutive;
4. Trebuie să existe în sistem o **unitate de calcul**, capabilă să efectueze operații **aritmetice și logice** asupra datelor (operanzilor) din memorie;
5. Este necesar un **mediu de ieșire**, prin intermediul căruia rezultatele sunt extrase de utilizator.

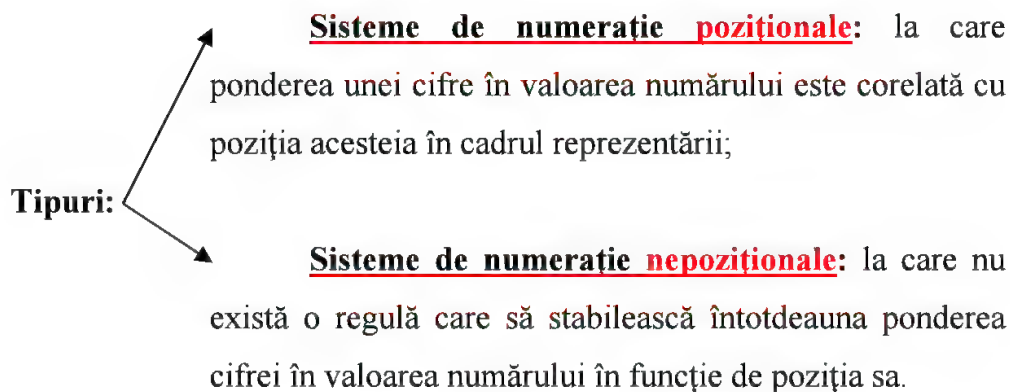


Observație: – În prezent, se utilizează aproape în exclusivitate numai mașini de calcul digitale (numerice).

2. Sisteme de numerație

Definiție: – Prin **sistem de numerație** se înțelege un ansamblu de reguli care precizează modul de utilizare a unei mulțimi de simboluri (numite **cifre**), pentru reprezentarea valorilor numerice.

– **Baza** unui sistem de numerație este dată de numărul cifrelor distincte utilizate.

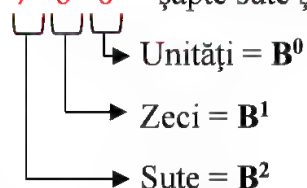


Exemple:

1. **Sistemul zecimal:** – folosește zece simboluri distincte;

- Mulțimea de simboluri: $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$;
- Baza are valoarea: $B = \text{zece}$;

– Este un sistem pozițional: $7 \ 6 \ 8 = \text{șapte sute șaiszeci și opt}$



2. **Sistemul roman:** – folosește șapte simboluri distincte;

- Mulțimea de simboluri: $S = \{I(\text{unu}), V(\text{cinci}), X(\text{zece}), L(\text{cincizeci}), C(\text{o sută}), D(\text{cinci sute}), M(\text{o mie})\}$;

– Baza are valoarea: $B = \text{șapte}$;

– Este un sistem nepozițional, deoarece simbolurile au întotdeauna aceeași pondere, indiferent de poziția lor;

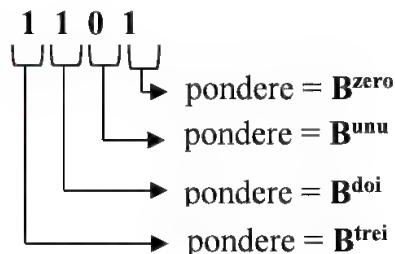
$$\begin{cases} IX = \text{unu înainte de zece (nouă)}; \\ XI = \text{unu după zece (unsprezece)}; \\ XII = \text{doi de unu după zece (doisprezece)} \end{cases}$$

CXXXII = o sută, trei de zece și doi după = o sută treizeci și doi.

3. **Sistemul binar:** – folosește doar două simboluri;

$$S = \{0,1\}; B = \text{doi};$$

- Este pozițional;



- Este sistemul folosit intern de toate mașinile digitale de calcul, fiind ușor de materializat practic:

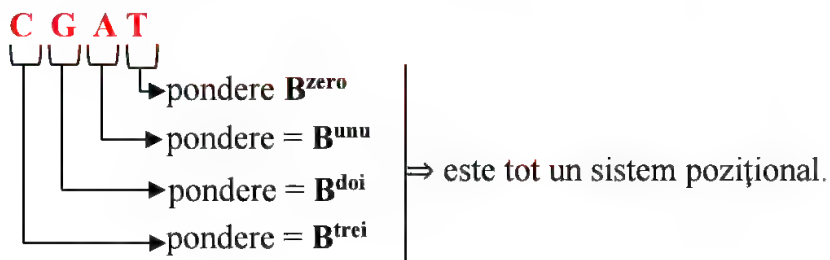
↙ cifra **0** = absență tensiune electrică;
↘ cifra **1** = prezență tensiune electrică.

4. **Sistemul cuaternar:** – folosește patru simboluri;

$$S = \{A, C, G, T\}; B = \text{patru};$$

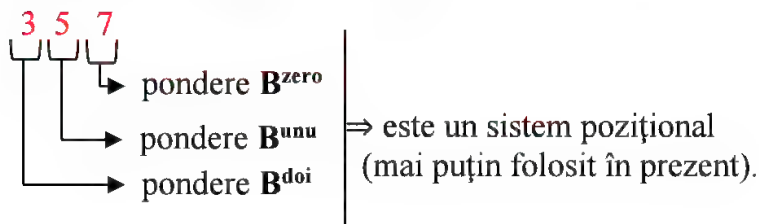
$$\begin{cases} A = \text{zero} = \text{ADENINĂ} \\ C = \text{unu} = \text{CITOZINĂ} \\ G = \text{doi} = \text{GUANINĂ} \\ T = \text{trei} = \text{TIMINĂ} \end{cases} \Rightarrow \text{poate fi considerat sistemul vieții}$$

deoarece este folosit pentru reprezentarea codurilor genetice în



5. **Sistemul octal:** – folosește opt simboluri distincte;

$$S = \{0,1,2,3,4,5,6,7\}; B = \text{opt};$$

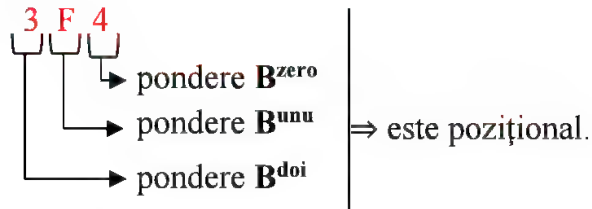


6. **Sistemul hexazecimal:** – folosește șaisprezece simboluri;

$S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A(\text{zece}), B(\text{unsprezece}),$

$C(\text{doisprezece}), D(\text{treisprezece}), E(\text{paisprezece}), F(\text{cincisprezece})\}$;

$B = \text{șaisprezece};$



– Se folosește pentru reprezentarea mai compactă în exteriorul mașinilor de calcul a valorilor numerice care intern sunt în sistemul binar.

Observatii: – Pentru oricare sistem de numerație pozițional, **valoarea bazei** se scrie în sistemul respectiv sub forma: $B = 10$ (cu două cifre, dar citite “unu zero” și nu “zece”);

– Se obișnuiește ca valoarea bazei (*scrisă anterior în cuvinte*) să fie reprezentată întotdeauna în sistemul de numerație zecimal și nu în cel de reprezentare al numărului; pentru evitarea confuziilor, valoarea bazei a fost scrisă în cuvinte și nu în sistemul zecimal.

Definiție: Într-un sistem de numerație pozițional de bază B , un număr cu valoarea N care are parte întreagă și parte fracționară separate prin virgulă se scrie sub forma:

$$N = (a_{n-1}a_{n-2} \dots a_1a_0, a_{-1}a_{-2} \dots a_{-(m-1)}a_{-m})_B$$

unde: $a_k \in \{0, 1, \dots, B - 1\}$ sunt cifrele numărului în baza B , iar

$$k = \overline{-m, n-1}$$

2.1. Trecerea unui număr dintr-o bază în alta

2.1.1. Trecerea din baza B în baza zece (în zecimal):

Valoarea în zecimal a unui număr N din baza B se determină prin *dezvoltarea polinomială după puterile bazei B* (sistemul fiind pozițional, se înmulțește valoarea fiecărei cifre cu ponderea dată de poziția ei):

$$\begin{aligned} N|_{10(\text{zece})} &= a_{n-1} \cdot B^{n-1} + a_{n-2} \cdot B^{n-2} + \dots + a_1 \cdot B^1 + a_0 \cdot B^0 + \\ &+ a_{-1} \cdot B^{-1} + a_{-2} \cdot B^{-2} + \dots + a_{-(m-1)} \cdot B^{-(m-1)} + a_{-m} \cdot B^{-m} \end{aligned}$$

$$N = \sum_{k=-m}^{n-1} a_k \cdot B^k$$

Exemple: – Calculul valorii zecimale pentru numerele folosite anterior;

$$\left\{ \begin{matrix} 768 \\ B = 10 \end{matrix} \right\} \Rightarrow N_{10} = 7 \cdot 10^2 + 6 \cdot 10^1 + 8 \cdot 10^0 = 7 \cdot 100 + 6 \cdot 10 + 8 \cdot 1 = 768;$$

$$\left\{ \begin{matrix} 1101 \\ B = 2 \end{matrix} \right\} \Rightarrow N_{10} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = \\ = 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 13$$

$$\left\{ \begin{matrix} \overset{\text{C}}{1} \overset{\text{G}}{2} \overset{\text{A}}{0} \overset{\text{T}}{3} \\ B = 4 \end{matrix} \right\} \Rightarrow N_{10} = 1 \cdot 4^3 + 2 \cdot 4^2 + 0 \cdot 4^1 + 3 \cdot 4^0 = \\ = 1 \cdot 64 + 2 \cdot 16 + 0 \cdot 4 + 3 \cdot 1 = 99;$$

$$\left\{ \begin{matrix} 357 \\ B = 8 \end{matrix} \right\} \Rightarrow N_{10} = 3 \cdot 8^2 + 5 \cdot 8^1 + 7 \cdot 8^0 = 3 \cdot 64 + 5 \cdot 8 + 7 = 239;$$

$$\left\{ \begin{matrix} 3F4 \\ B = 16 \end{matrix} \right\} \Rightarrow N_{10} = 3 \cdot 16^2 + 15 \cdot 16^1 + 4 \cdot 16^0 = 3 \cdot 256 + 15 \cdot 16 + 4 = 1012;$$

Observații: – Valoarea maximă care poate fi reprezentată direct (fără codificare) când se folosesc x cifre în baza B este: $(N)_{max} = B^x - 1$ (când toate cifrele au valoarea $B-1$);

– Aceeași valoare se poate reprezenta în baza R folosind un număr de y cifre: $(N)_{max} = R^y - 1 = B^x - 1 \Leftrightarrow R^y = B^x \Rightarrow y = x \cdot \log_R B$;

– Dacă $B = 10$ și $R = 2$ avem: $y = x \cdot \log_2 10 = \frac{x}{\log_{10} 2} \cong x \cdot 3,32 \Rightarrow$ în baza 2 sunt necesare de 3,32 ori mai multe cifre decât în baza 10 pentru a reprezenta aceeași valoare numerică. Dacă raportul dintre numărul necesar de cifre în cele două baze ar fi fost întreg, trecerea reprezentării dintr-o bază în cealaltă s-ar fi putut face direct, pe grupuri independente de cifre, fără a mai considera valoarea numărului în ansamblu.

– Numerele cu valori între 0 și 16 reprezentate în bazele 10 (zecimal), 2 (binar), 4 (cuaternar), 8 (octal), 16 (hexazecimal) se pot obține simplu prin incrementare (numărare directă cu adunarea unei unități), dacă se ține cont că la atingerea valorii bazei se adună o unitate la cifra de pe rangul următor:

Valoare numerică $N \downarrow$	$N _{10}$ (zecimal)		$N _2$ (binar)					$N _4$ (cuaternar)			$N _8$ (octal)		$N _{16}$ (hexazecimal)	
Ponderi cifre $\rightarrow$	10^1	10^0	2^4	2^3	2^2	2^1	2^0	4^2	4^1	4^0	8^1	8^0	16^1	16^0
0	0	0	0	0	0	0	0	A	A	A	0	0	0	0
1	0	1	0	0	0	0	1	A	A	C	0	1	0	1
2	0	2	0	0	0	1	0	A	A	G	0	2	0	2
3	0	3	0	0	0	1	1	A	A	T	0	3	0	3
4	0	4	0	0	1	0	0	A	C	A	0	4	0	4
5	0	5	0	0	1	0	1	A	C	C	0	5	0	5
6	0	6	0	0	1	1	0	A	C	G	0	6	0	6
7	0	7	0	0	1	1	1	A	C	T	0	7	0	7
8	0	8	0	1	0	0	0	A	G	A	1	0	0	8
9	0	9	0	1	0	0	1	A	G	C	1	1	0	9
10	1	0	0	1	0	1	0	A	G	G	1	2	0	A
11	1	1	0	1	0	1	1	A	G	T	1	3	0	B
12	1	2	0	1	1	0	0	A	T	A	1	4	0	C
13	1	3	0	1	1	0	1	A	T	C	1	5	0	D
14	1	4	0	1	1	1	0	A	T	G	1	6	0	E
15	1	5	0	1	1	1	1	A	T	T	1	7	0	F
16	1	6	1	0	0	0	0	C	A	A	2	0	1	0

Observații: – Valorile înconjurate se obțin prin transport la cifra cu ponderea imediat superioară, datorită atingerii valorii bazei.

– Pentru verificare, se poate urmări pe verticală (coloană) valoarea fiecărei cifre cu ponderea B^k : aceasta trebuie să se repete de B^k ori înainte de a trece la valoarea următoare.

2.1.2. Trecerea unui număr întreg din zecimal în baza B:

$$N = (a_{n-1}a_{n-2} \dots a_1a_0)_B; \text{ cu: } B \in \mathbb{N}^* \text{ și } \begin{cases} a_i \in \mathbb{N}; \\ a_i \leq B - 1. \end{cases}$$

↳ valoarea zecimală este:

$$N|_{10} = (a_{n-1} \cdot B^{n-1} + a_{n-2} \cdot B^{n-2} + \dots + a_1 \cdot B^1 + a_0 \cdot B^0)$$

– De această dată se cunoaște valoarea zecimală $N|_{10}$ (notată mai departe cu N simplu) și se dorește obținerea coeficienților polinomului (a cifrelor) a_i .

$$N = B \cdot (a_{n-1} \cdot B^{n-2} + a_{n-2} \cdot B^{n-3} + \dots + a_1) + a_0 \mid \Rightarrow$$

factor comun; notat cu Q_1 ;

$$\Rightarrow N = B \cdot Q_1 + a_0 \mid \Rightarrow \frac{N}{B} = Q_1, \text{rest } a_0;$$

$$Q_1 = B \cdot (a_{n-1} \cdot B^{n-3} + a_{n-2} \cdot B^{n-4} + \dots + a_2) + a_1 \mid \Rightarrow$$

notat cu Q_2 ;

$$\Rightarrow Q_1 = B \cdot Q_2 + a_1 \mid \Rightarrow \frac{Q_1}{B} = Q_2, \text{rest } a_1.$$

– Pentru obținerea coeficienților, se împarte valoarea numărului N la valoarea bazei (ambele considerate în sistemul zecimal), în mod repetat, de **n ori**, până la obținerea lui **a_{n-1}** (cu câtul zero);

– La fiecare împărțire, se obține câte un coeficient dat de valoarea **restului** împărțirii; coeficienții se obțin în ordine inversă.

Exemplu: $B = 2$; $N|_{10} = 739$;

739 : 2 =		N=1 0 1 1 1 0 0 0 1 1 $ _2$									
369	rest 1	ORDINE A COEF F I C I E N T I I									
184	rest 1										
92	rest 0										
46	rest 0										
23	rest 0										
11	rest 1										
5	rest 1										
2	rest 1										
1	rest 0										
câtul zero	0 rest 1										

– Metodă particulară pentru baza $B=2$: – Se găsește cea mai mare putere a lui 2 care se poate scădea din numărul respectiv, apoi cea mai mare putere care se poate scădea din diferența rămasă, în mod repetitiv până la puterea 2^1 , când se obține $\mathbf{a_0}$ (metoda puterilor).

$2^0 =$	1	$2^9 =$	512
$2^1 =$	2	$2^{10} =$	1024
$2^2 =$	4	$2^{11} =$	2048
$2^3 =$	8	$2^{12} =$	4096
$2^4 =$	16	$2^{13} =$	8192
$2^5 =$	32	$2^{14} =$	16384
$2^6 =$	64	$2^{15} =$	32768
$2^7 =$	128	$2^{16} =$	65536
$2^8 =$	256		

- Metoda este practică până la puterea 2^{16} , deoarece valorile puterilor sunt frecvent întâlnite în tehnica de calcul și *se memorează*.

Exemplu:

restul coeficienților sunt zero:

↓

$$\begin{array}{cccccccccc} \mathbf{a_9} & \mathbf{a_8} & \mathbf{a_7} & \mathbf{a_6} & \mathbf{a_5} & \mathbf{a_4} & \mathbf{a_3} & \mathbf{a_2} & \mathbf{a_1} & \mathbf{a_0} \\ (1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1)_2 \end{array}$$

$$\left\{ \begin{array}{l} 739 - \boxed{512}_{2^9} = 227 \mid \Rightarrow a_9 = 1 \\ \downarrow \\ 227 - \boxed{128}_{2^7} = 99 \mid \Rightarrow a_7 = 1 \\ \downarrow \\ 99 - \boxed{64}_{2^6} = 35 \mid \Rightarrow a_6 = 1 \\ \downarrow \\ 35 - \boxed{32}_{2^5} = 3 \mid \Rightarrow a_5 = 1 \\ \downarrow \\ 3 - \boxed{2}_{2^1} = 1 \mid \Rightarrow a_1 = 1 \end{array} \right. \quad \begin{array}{l} a_0 = 1 \end{array}$$

2.1.3. Trecerea unui număr subunitar din zecimal în baza B:

$$N = (0, a_{-1}a_{-2} \dots a_{-(m-1)}a_{-m})_B; \text{ cu: } \begin{cases} B \in \mathbb{N}^*; \\ a_i \in \mathbb{N}; \\ a_i \leq B - 1. \end{cases}$$

↳ valoarea zecimală este:

$$N|_{10} = a_{-1} \cdot B^{-1} + a_{-2} \cdot B^{-2} + \dots + a_{-(m-1)} \cdot B^{-(m-1)} + a_{-m} \cdot B^{-m};$$

– Se cunoaște $N|_{10}$ notată simplu cu N și se caută a_i .

$$N = B^{-1} \cdot (a_{-1} + a_{-2} \cdot B^{-1} + \dots + a_{-(m-1)} \cdot B^{-(m-2)} + a_{-m} \cdot B^{-(m-1)}) \Rightarrow$$

$$\Rightarrow N \cdot B = \boxed{a_{-1}} + \underbrace{a_{-2} \cdot B^{-1} + \dots + a_{-m} \cdot B^{-(m-1)}}_{\text{notat cu } F_1 < 1} \Rightarrow N \cdot B = a_{-1} + F_1 \Rightarrow$$

$$\Rightarrow \boxed{a_{-1} = [N \cdot B]_{\text{partea întreagă}}};$$

$$F_1 \cdot B = \boxed{a_{-2}} + \underbrace{a_{-3} \cdot B^{-1} + \dots + a_{-m} \cdot B^{-(m-2)}}_{\text{notat cu } F_2}$$

$$\Rightarrow \boxed{a_{-2} = [F_1 \cdot B]_{\text{partea întreagă}}};$$

– Pentru obținerea coeficienților, se înmulțește valoarea numărului subunitar N cu valoarea bazei (ambele considerate în sistemul zecimal), apoi procesul se repetă cu părțile fracționare intermediare până când se obține o parte fracționară $F_i = \text{zero}$ sau se atinge precizia de reprezentare dorită.

– La fiecare înmulțire, se obține câte un coeficient dat de valoarea **părții întregi** a înmulțirii; coeficienții se obțin în ordine directă.

Exemple:

$$B = 2; N|_{10} = 0,625 = \frac{625}{1000} = \frac{5}{8};$$

$$\left\{ \begin{array}{l} 0,625 \cdot 2 = \boxed{1},25 \\ \quad \downarrow \\ 0,25 \cdot 2 = \boxed{0},5 \\ \quad \downarrow \\ 0,5 \cdot 2 = \boxed{1},0 \\ \quad \text{zero} \end{array} \right. \begin{array}{l} \text{Ordine} \\ \text{coeficienți} \end{array}$$

$$N = 0,101|_2$$

$$B = 2; N|_{10} = 0,78 = \frac{78}{100} = \frac{39}{50};$$

$0,78 \cdot 2 =$		O R D I N E A C O E F I C I E N T I L O R
0,56	parte întreagă 1	
0,12	parte întreagă 1	
0,24	parte întreagă 0	
0,48	parte întreagă 0	
0,96	parte întreagă 0	
0,92	parte întreagă 1	
0,84	parte întreagă 1	
0,68	parte întreagă 1	
⋮	⋮	
⋮	⋮	
(continuă la infinit)		

$$N = 0,11000111 \dots |_2$$

(formă **periodică** cu o infinitate de cifre !)

Observații: – Un **număr rațional** (care se poate exprima exact prin raportul a două valori întregi) are o formă finită într-o bază dacă toți factorii primi din descompunerea numitorului fracției reduse (simplificate) sunt și factori ai bazei respective, altfel va avea o **formă infinită periodică**. Dacă numitorul fracției reduse are cel puțin un factor prim comun cu baza, partea periodică este precedată de o parte tranzitorie cu lungime finită.

– Cu cât baza sistemului de numerație folosit are mai mulți factori primi, cu atât se pot exprima mai multe numere raționale sub formă finită. În primul exemplu, numitorul fracției reduse are doar valoarea 2 ca factor prim, care este și factor al **bazei doi**; în următorul exemplu însă, numitorul are în plus și valoarea 5 ca factor prim, care nu mai este factor al **bazei doi**; exprimarea valorii numărului este finită în **baza zece**, care are și factorul prim 5.

– Un **număr irațional** (care nu se poate exprima exact prin raportul a două valori întregi) va rămâne sub aceeași formă **infinită și neperiodică** în toate bazele, cu excepția notabilă când se folosește ca bază propria valoare și exprimarea sa devine “10”.

2.1.4. Trecerea unui număr cu virgulă fixă**din zecimal într-o bază oarecare B:**

– Se realizează prin prelucrarea separată a părții întregi și părții fracționare, iar apoi se concatenează rezultatul în jurul virgulei.

Exemplu: $B = 2$; $N|_{10} = 21,125$;

$$\begin{array}{r|l} 21 : 2 & \\ \hline 10 & 1 \\ 5 & 0 \\ 2 & 1 \\ 1 & 0 \\ 0 & 1 \\ \hline \text{END} & \end{array}$$

$$\begin{array}{r|l} 0,125 \cdot 2 & \\ \hline 0,25 & 0 \\ 0,5 & 0 \\ 1 & 1 \\ \hline \text{END} & \end{array}$$

$$N = 10101,001_2$$

Verificare: $N|_{10} = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 16 + 4 + 1 + \frac{1}{8} = 21 + 0,125 = 21,125.$

2.1.5. Trecerea unui număr din baza R în baza B**când B este o putere a lui R:**

$$N|_R \rightarrow N|_B; \text{ cu: } B = R^P;$$

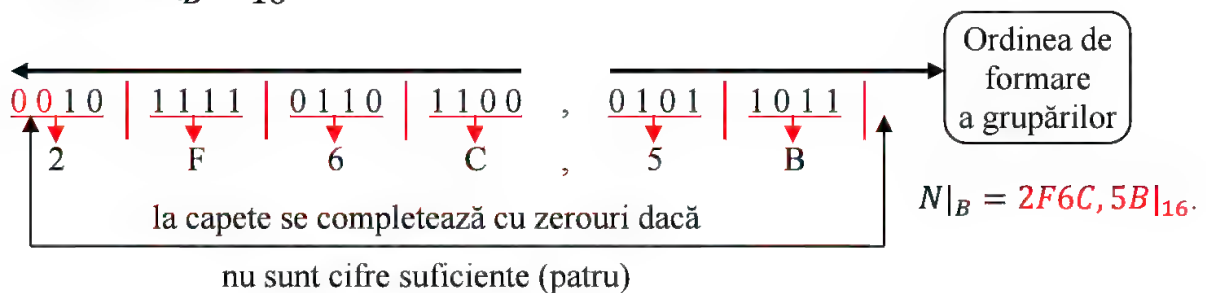
– Dacă N are y cifre în baza R , el va avea x cifre în baza B , conform relației anterioare: $R^y = B^x = (R^P)^x \Rightarrow y = P \cdot x \Leftrightarrow x = \frac{y}{P} \Rightarrow$ în baza mai mare B , numărul va avea **exact** de P ori mai puține cifre.

– Se grupează cifrele numărului din baza R , în perechi de câte P , de la virgulă spre stânga pentru partea întreagă, respectiv de la virgulă spre dreapta pentru partea fracționară și se substituie grupările astfel formate cu cifra echivalentă lor din baza B (cifra care are aceeași valoare ca gruparea).

Cazuri particulare:

- baza 2 $\rightarrow$ baza 4 $\mid \Rightarrow \boxed{R = 2} \xrightarrow{N} \boxed{B = 4} = 2^2 \Rightarrow$ se grupează câte 2 cifre binare și se înlocuiesc cu una cuaternară;
- baza 2 $\rightarrow$ baza 8 $\mid \Rightarrow \boxed{R = 2} \xrightarrow{N} \boxed{B = 8} = 2^3 \Rightarrow$ se grupează câte 3 cifre binare și se înlocuiesc cu una octală;
- baza 2 $\rightarrow$ baza 16 $\mid \Rightarrow \boxed{R = 2} \xrightarrow{N} \boxed{B = 16} = 2^4 \Rightarrow$ se grupează câte 4 cifre binare și se înlocuiesc cu una hexazecimală;
- baza 4 $\rightarrow$ baza 16 $\mid \Rightarrow \boxed{R = 4} \xrightarrow{N} \boxed{B = 16} = 4^2 \Rightarrow$ se grupează câte 2 cifre cuaternare și se înlocuiesc cu una hexazecimală.

Exemplu: $\begin{cases} R = 2 \\ B = 16 \end{cases}$; $N|_R = 10111101101100,01011011|_2$;



Observație: – Este necesară cunoașterea reprezentărilor numerelor cu valori între 0 și 15 în toate bazele (conform tabelului prezentat anterior).

2.1.6. Trecerea unui număr din baza B în baza R **când B este o putere a lui R:**

$N|_B \rightarrow N|_R$; cu: $B = R^P$;

– Dacă N are x cifre în baza B , el va avea y cifre în baza R , conform relației anterioare: $y = x \cdot \log_R B = \frac{x}{\log_B R} = x \cdot \log_R(R^P) = x \cdot P \Rightarrow y = P \cdot x \Rightarrow$ în baza R vor fi **exact** P cifre pentru fiecare din cifrele bazei B .

– Se substituie fiecare cifră a numărului din baza B cu câte o pereche de P cifre care reprezintă echivalentul valorii cifrei inițiale în noua bază.

Cazuri particulare:

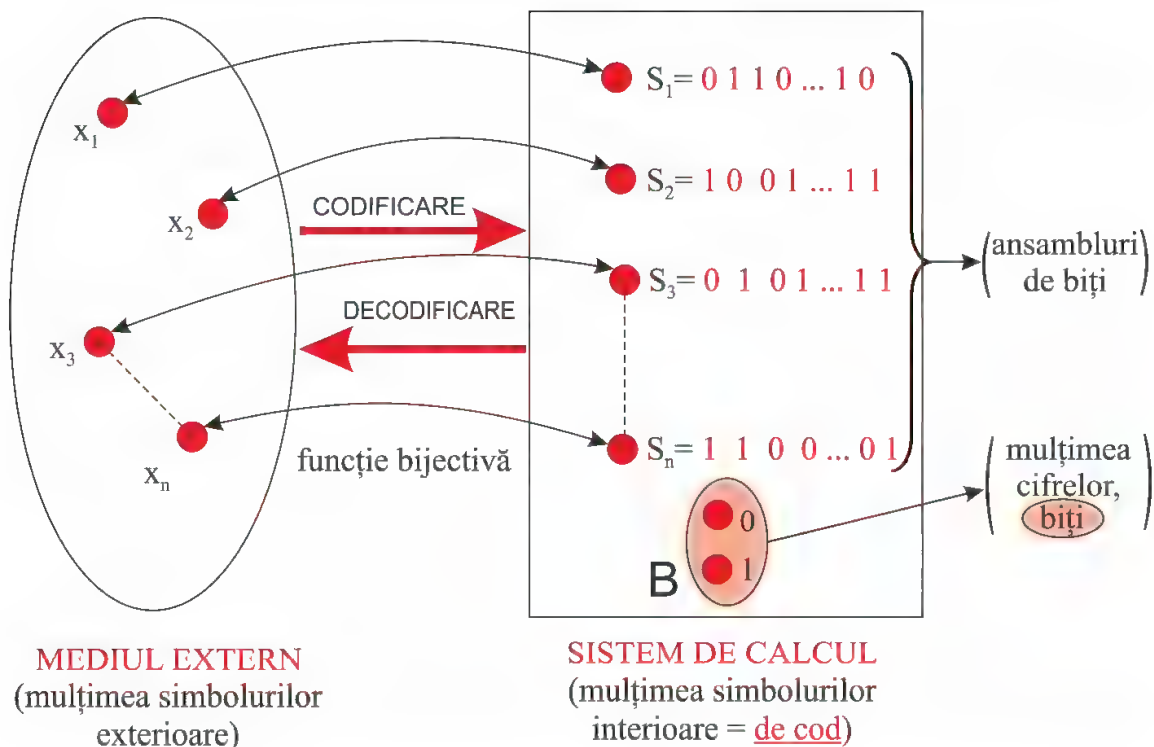
- baza 16 $\rightarrow$ baza 4 $\Rightarrow$ $B = 16 = 4^2 \rightarrow R = 4$ $\Rightarrow$ se înlocuiește fiecare cifră hexazecimală cu o pereche de 2 cifre cuaternare;
- baza 16 $\rightarrow$ baza 2 $\Rightarrow$ $B = 16 = 2^4 \rightarrow R = 2$ $\Rightarrow$ se înlocuiește fiecare cifră hexazecimală cu o pereche de 4 cifre binare;
- baza 8 $\rightarrow$ baza 2 $\Rightarrow$ $B = 8 = 2^3 \rightarrow R = 2$ $\Rightarrow$ se înlocuiește fiecare cifră octală cu o pereche de 3 cifre binare;
- baza 4 $\rightarrow$ baza 2 $\Rightarrow$ $B = 4 = 2^2 \rightarrow R = 2$ $\Rightarrow$ se înlocuiește fiecare cifră cuaternară cu o pereche de 2 cifre binare.

Exemplu: $\begin{cases} B = 16 \\ R = 2 \end{cases}; \quad N|_B = 3AB,5FE|_{16};$

$\begin{array}{cccccc} \text{3} & \text{A} & \text{B} & , & \text{5} & \text{F} & \text{E} \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow \\ \overline{0011} & \overline{1010} & \overline{1011} & , & \overline{0101} & \overline{1111} & \overline{1110} \end{array}$

$\Rightarrow$ $N|_R = 001110101011,010111111110|_2.$

3. Reprezentarea informației în sistemele de calcul

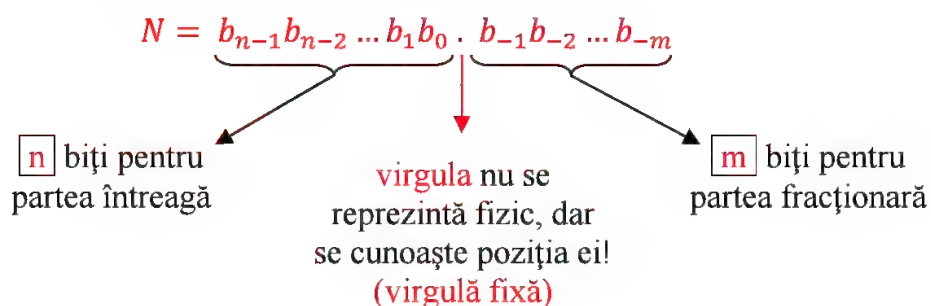


Definiție: – Schimbarea formei de reprezentare a informației poartă denumirea de codificare / decodificare.

– Pentru a putea lucra cu informațiile din mediul exterior (valori numerice în diferite baze sau litere alfabetice), un sistem de calcul are nevoie de reprezentarea acestora în forma internă, ca ansambluri de biți organizate după anumite reguli, denumite simboluri de cod.

3.1. Reprezentarea numerelor fără semn în virgulă fixă

– Valorile numerice fără semn se codifică în calculator sub formă identică cu reprezentarea externă în baza doi; cifrele numărului în baza doi devin biți de cod:



Domeniul de valori reprezentabile:

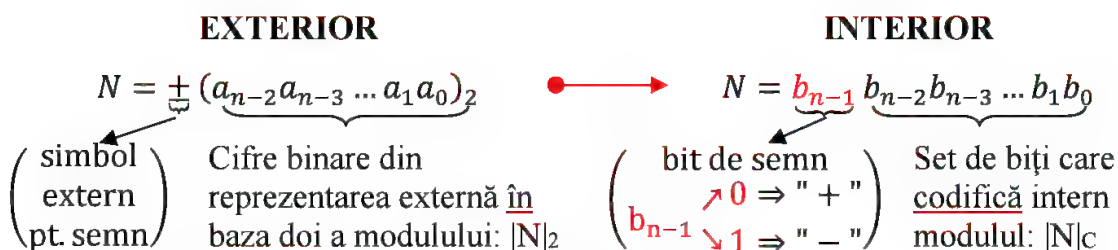
$$\left\{ \begin{array}{l}
 \text{– Partea întreagă: } 0 \leq N_i \leq 2^n - 1 \\
 \quad \text{când toți cei } [n] \text{ biți sunt 1} \\
 \text{– Partea fracționară: } 0 \leq N_f \leq \frac{2^m - 1}{2^m} \\
 \quad \text{când toți cei } [m] \text{ biți sunt 1}
 \end{array} \right. \Rightarrow \begin{array}{l}
 0 \leq N_i + N_f \leq 2^n - 1 + \\
 + \frac{2^m - 1}{2^m} = 2^n - \frac{1}{2^m} \quad | \Rightarrow \\
 \Rightarrow 0 \leq N \leq 2^n - 2^{-m}
 \end{array}$$

– Dacă numărul se aduce la forma subunitară prin împărțire cu 2^n , virgula va fi totdeauna plasată în fața bitului cel mai semnificativ din cadrul codului:

$$\begin{cases} N' = \frac{N}{2^n} = 0.b_{n-1}b_{n-2} \dots b_1b_0b_{-1} \dots b_{-(m-1)}b_{-m}; \\ 0 \leq N' \leq 1 - 2^{-(n+m)} \end{cases}$$

3.2. Reprezentarea numerelor întregi cu semn

– Atunci când valoarea numerică externă are semn, acesta trebuie și el codificat în cadrul ansamblului simbolului intern: se alocă bitul din poziția cea mai semnificativă a codului binar pentru reprezentarea semnului.



$$\pm |N|_2 \Leftrightarrow [b_{n-1}|N|_c]$$

După modul cum se precizează biții care codifică intern modulul ($|N|_2 \Leftrightarrow |N|_C$), există 3 variante:

$$\pm|N|_2 = \begin{cases} +|N|_2 \Leftrightarrow [0|N|_2]; \\ \text{(pt. numere pozitive se} \\ \text{păstrează cifrele binare)} \\ -|N|_2 \Leftrightarrow [1|N|_C] = \begin{cases} [1|N|_2] \Rightarrow \text{mărime și semn}; \\ [1|N|_{C_1}] \Rightarrow \text{cod complementar față de 1}; \\ [1|N|_{C_2}] \Rightarrow \text{cod complementar față de 2}. \end{cases} \\ \text{(pt. numerele negative} \\ \text{cifrele binare se modifică} \\ \text{prin codificare)} \end{cases}$$

3.2.1. Reprezentarea în mărime și semn:

– Pentru ambele valori (pozitive și negative) se preiau chiar cifrele binare din reprezentarea în baza doi a modulului:

$$b_k = a_k, k = \overline{0, n-2}.$$

$$N = \begin{cases} \overset{MS}{+|N|_2 \Leftrightarrow [0|N|_2]} = 0 a_{n-2} a_{n-3} \dots a_1 a_0; \text{ pt. } N \geq 0; \\ \overset{MS}{-|N|_2 \Leftrightarrow [1|N|_2]} = 1 a_{n-2} a_{n-3} \dots a_1 a_0; \text{ pt. } N \leq 0; \end{cases}$$

$$\left. \begin{aligned} |N|_{10} &= b_{n-2} \cdot 2^{n-2} + b_{n-3} \cdot 2^{n-3} + \dots + b_1 \cdot 2 + b_0 \\ N &= \begin{cases} -|N|_{10}; \text{ pentru } b_{n-1} = 1 \\ +|N|_{10}; \text{ pentru } b_{n-1} = 0 \end{cases} = (-1)^{b_{n-1}} \cdot |N|_{10} \end{aligned} \right\} \Rightarrow N = (-1)^{b_{n-1}} \cdot \sum_{k=0}^{n-2} b_k \cdot 2^k$$

Dacă se cunoaște reprezentarea internă, valoarea zecimală externă se calculează considerând efectul bitului de semn asupra modulului:

Domeniul de reprezentare se determină cu formula valorii zecimale externe:

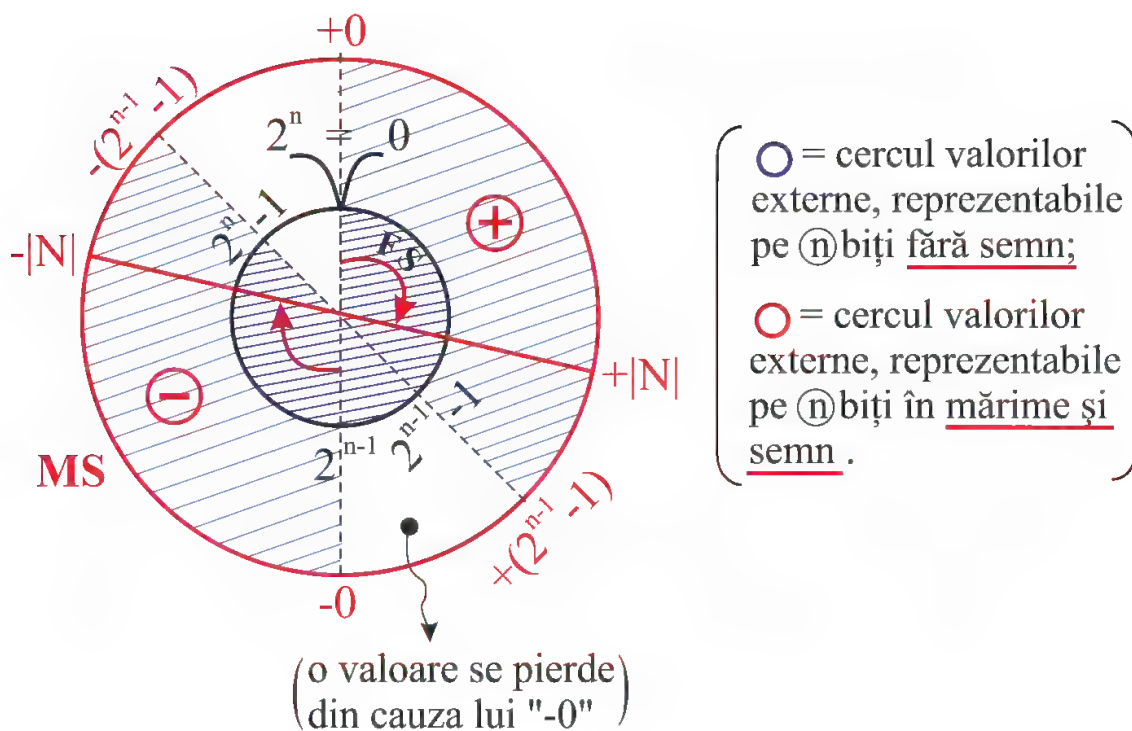
$$\left. \begin{aligned} N_{min} &= (-1)^{\boxed{1}} \cdot \sum_{k=0}^{n-2} \boxed{1} \cdot 2^k = -(2^{n-1} - 1) \\ N_{max} &= (-1)^{\boxed{0}} \cdot \sum_{k=0}^{n-2} \boxed{1} \cdot 2^k = +(2^{n-1} - 1) \end{aligned} \right\} \Rightarrow -(2^{n-1} - 1) \leq N \leq 2^{n-1} - 1$$

$\rightarrow$ toate cifrele binare $b_k=1; k = \overline{0, n-2}$;
 $\rightarrow$ bitul de semn $b_{n-1} = \begin{cases} 1; \text{ pentru minim;} \\ 0; \text{ pentru maxim.} \end{cases}$

Observații: – Codificarea în mărime și semn conduce la două reprezentări distincte pentru valoarea zero:

$$0 = \begin{cases} +0 \Leftrightarrow \overset{MS}{0} \underbrace{00 \dots 00}_{(n-1) \text{ biți}} \\ -0 \Leftrightarrow \overset{MS}{1} \underbrace{00 \dots 00}_{(n-1) \text{ biți}} \end{cases} \Rightarrow \begin{array}{l} \text{Se pierde un cod și se} \\ \text{complică construcția} \\ \text{unității de calcul} \end{array}$$

– Dacă toate valorile care ies în afara domeniului de reprezentare sunt considerate **modulo $2^{nr. \text{ de biți}}$** , atunci valorile reprezentabile se pot plasa pe câte un cerc:



3.2.2. Reprezentarea în cod complementar față de 1:

Definiție: – Pentru un număr A pozitiv reprezentat în baza B folosind n cifre (simboluri) se definește codul complementar față de baza micșorată cu o unitate prin:

$$C_{B-1}(A) = (B^n - 1) - A \mid \xrightarrow{B=2} C_1(A) = (2^n - 1) - A.$$

– Codul complementar C_1 al unui singur bit este dat de valoarea inversată a bitului:

$$n = 1 \Rightarrow C_1(b_k) = (2^1 - 1) - b_k = \boxed{1 - b_k \text{ notat } \overline{b_k}}$$

$$\left(b_k = \begin{cases} 0 & | \Rightarrow \overline{b_k} = 1; \\ 1 & | \Rightarrow \overline{b_k} = 0; \end{cases} \right)$$

– Reprezentarea în cod complementar față de 1 a numerelor cu semn se face prin inversarea cifrelor binare din forma în baza doi a modulului, **pentru valori negative**:

$$b_k = \begin{cases} a_k; & \text{pt. } N \geq 0; \\ \overline{a_k}; & \text{pt. } N \leq 0; \end{cases} \text{ cu: } k = \overline{0, n-2}.$$

$$N = \begin{cases} +|N|_2 \xrightarrow{C_1} [0|N|_2] = 0 a_{n-2} a_{n-3} \dots a_1 a_0; & \text{pt. } N \geq 0; \\ -|N|_2 \xrightarrow{C_1} [1|N|_{C_1}] = 1 \overline{a}_{n-2} \overline{a}_{n-3} \dots \overline{a}_1 \overline{a}_0; & \text{pt. } N \leq 0; \end{cases}$$

Demonstrație: $|N|_{C_1} \stackrel{\text{def}}{=} (2^{n-1} - 1) - |N_2| =$

$$= \underbrace{(1 \ 1 \ \dots \ 1 \ 1)}_{(n-1) \text{ biți}}_2 - \underbrace{(a_{n-2} a_{n-3} \dots a_1 a_0)}_{(n-1) \text{ biți}}_2 =$$

$$= [(1 - a_{n-2}), (1 - a_{n-3}), \dots, (1 - a_1), (1 - a_0)]_2 = \overline{a}_{n-2} \overline{a}_{n-3} \dots \overline{a}_1 \overline{a}_0.$$

(scăderea se face bit cu bit, fără împrumut)

– Dacă se cunoaște reprezentarea internă, valoarea zecimală externă se determină în funcție de bitul de semn, cu sau fără recomplementare față de 1:

$$N = \begin{cases} -(|N|_{C_1})_{10}; & \text{pt. } b_{n-1} = 1 \\ +|N|_{10}; & \text{pt. } b_{n-1} = 0 \end{cases} \left(|N|_{C_1} \right)_{10} = (2^{n-1} - 1) - |N|_{10} \Rightarrow N = \begin{cases} -[1 \cdot (2^{n-1} - 1) - |N|_{10}] & \text{pt. } b_{n-1} = 1 \\ -[0 \cdot (2^{n-1} - 1) - |N|_{10}] & \text{pt. } b_{n-1} = 0 \end{cases} \Rightarrow$$

$$\Rightarrow \boxed{N = -b_{n-1} \cdot (2^{n-1} - 1) + \sum_{k=0}^{n-2} b_k \cdot 2^k}$$

$\Rightarrow$ metoda (I.), cu definiția $\Rightarrow$

$$\Rightarrow N = \begin{cases} + \sum_{k=0}^{n-2} b_k \cdot 2^k; & \text{pt. } b_{n-1} = 0 \\ - \left[1 \cdot \boxed{\sum_{k=0}^{n-2} 2^k} - \sum_{k=0}^{n-2} b_k \cdot 2^k \right]; & \text{pt. } b_{n-1} = 1 \end{cases} =$$

$$\begin{aligned}
&= \begin{cases} + \sum_{k=0}^{n-2} b_k \cdot 2^k \\ - \sum_{k=0}^{n-2} (1 - b_k) \cdot 2^k \end{cases} = \begin{cases} + [0 \cdot 2^{n-1} + \sum_{k=0}^{n-2} b_k \cdot 2^k]; \text{ pt. } b_{n-1} = 0 \\ - [0 \cdot 2^{n-1} + \sum_{k=0}^{n-2} \bar{b}_k \cdot 2^k]; \text{ pt. } b_{n-1} = 1 \Leftrightarrow \bar{b}_{n-1} = 0 \end{cases} \\
&= \begin{cases} + [\underbrace{b_{n-1}}_{=0} \cdot 2^{n-1} + \sum_{k=0}^{n-2} b_k \cdot 2^k] \\ - [\underbrace{\bar{b}_{n-1}}_{=0} \cdot 2^{n-1} + \sum_{k=0}^{n-2} \bar{b}_k \cdot 2^k] \end{cases} = \begin{cases} + \sum_{k=0}^{n-1} b_k \cdot 2^k; \text{ pt. } b_{n-1} = 0; \\ - \sum_{k=0}^{n-1} \bar{b}_k \cdot 2^k; \text{ pt. } b_{n-1} = 1; \end{cases} \\
&\quad \text{metoda (II.), prin recomplementare.}
\end{aligned}$$

Observație: – Se poate include și bitul de semn la recomplementare, acesta conducând la o valoare zero (nr. pozitive) sau unu complementat = zero (la nr. negative).

Domeniul de reprezentare se determină cu formula valorii zecimale externe (metoda I.):

$$\begin{aligned}
N_{min} &= -\boxed{1} \cdot (2^{n-1} - 1) + \sum_{k=0}^{n-2} \boxed{0} \cdot 2^k = -(2^{n-1} - 1) \\
N_{max} &= -\boxed{0} \cdot (2^{n-1} - 1) + \sum_{k=0}^{n-2} \boxed{1} \cdot 2^k = +(2^{n-1} - 1) \\
&\Rightarrow -(2^{n-1} - 1) \leq N \leq 2^{n-1} - 1;
\end{aligned}$$

Exemple: $n = 6$ biți; $N = +12 \xrightarrow{C_1} ?$

$$|N|_2 = \left| \overset{b_5}{\underbrace{+}_{b_5=0}} 1100 \right| = 1100 \Rightarrow N_{C_1} = \begin{matrix} & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ \boxed{0} & \boxed{0} & 1 & 1 & 0 & 0 \end{matrix}$$

bit de semn bit zero pt. completarea
modulului la 5 biți

$$\begin{aligned}
N &= -b_5 \cdot (2^5 - 1) + \sum_{k=0}^4 b_k \cdot 2^k = -0 \cdot (32 - 1) + 0 \cdot 2^4 + 1 \cdot 2^3 + \\
&+ 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 8 + 4 = \boxed{+12} \text{ (valoarea externă cu formula generală);}
\end{aligned}$$

$n = 6$ biți; $N = -27 \xrightarrow{C_1} ?$

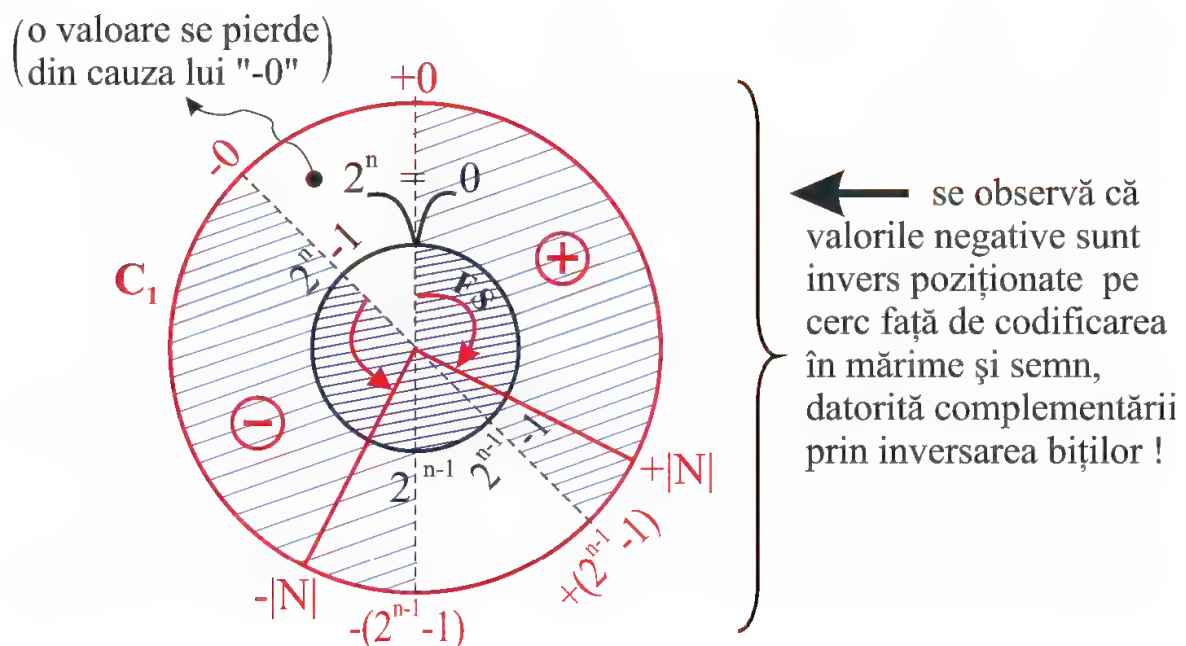
$$|N|_2 = \left| \overset{b_5}{\underbrace{-}_{b_5=1}} 11011 \right| = 11011 \Rightarrow N_{C_1} = \begin{matrix} & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ 1 & \bar{1} & \bar{1} & \bar{0} & \bar{1} & \bar{1} & \end{matrix} = \boxed{100100};$$

$$\begin{aligned}
 N &= -\sum_{k=0}^5 \bar{b}_k \cdot 2^k = -[\bar{1} \cdot 2^5 + \bar{0} \cdot 2^4 + \bar{0} \cdot 2^3 + \bar{1} \cdot 2^2 + \bar{0} \cdot 2^1 + \bar{0} \cdot 2^0] = \\
 &= -[0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0] = -[16 + 8 + 2 + 1] = \\
 &= \boxed{-27} \text{ (valoarea externă prin recomplementare pe toți biții).}
 \end{aligned}$$

Observații: – În reprezentarea C_1 există tot două coduri pentru valoarea zero:

$$\begin{aligned}
 0 &= \begin{cases} +0 \overset{C_1}{\underbrace{0 \ 0 \ 0 \dots 0 \ 0}}_{(n-1) \text{ biți}}; \\ -0 \overset{C_1}{\underbrace{1 \ 1 \ 1 \dots 1 \ 1}}_{(n-1) \text{ biți}}; \end{cases}
 \end{aligned}$$

– Reprezentarea pe cerc modulo 2^n a codurilor FS (fără semn) și C_1 (cod complementar față de 1):



3.2.3. Reprezentarea în cod complementar față de 2:

Definiție: – Pentru un număr pozitiv A reprezentat în baza B folosind $\boxed{n}$ cifre (simboluri), se definește codul complementar față de bază prin:

$$C_B(A) = B^n - A \mid \xRightarrow{B=2} C_2(A) = 2^n - A.$$

– Codul complementar față de bază se obține din codul complementar față de baza micșorată cu o unitate prin adunarea valorii 1:

$$C_{B-1}(A) = (B^n - 1) - A = (B^n - A) - 1 = C_B(A) - 1 \Rightarrow C_B(A) = C_{B-1}(A) + 1.$$

– Reprezentarea în cod complementar față de 2 a numerelor cu semn se face prin complementarea față de 1 a modului la care se adaugă valoarea 1, **pentru numerele externe negative**:

$$b_k = \begin{cases} a_k & \text{pt. } N \geq 0; \\ \tilde{a}_k & \text{pt. } N \leq 0; \end{cases} \text{ cu: } k = \overline{0, n-2}.$$

(notație pentru biții din C_2)

$$N = \begin{cases} +|N|_2 \xrightarrow{C_2} [0|N|_2] = 0 a_{n-2} a_{n-3} \dots a_1 a_0; \text{ pt. } N \geq 0; \\ -|N|_2 \xrightarrow{C_2} [1|N|_2] = 1 \tilde{a}_{n-2} \tilde{a}_{n-3} \dots \tilde{a}_1 \tilde{a}_0; \text{ pt. } N \leq 0; \end{cases}$$

Demonstratie:

$$\begin{aligned} |N|_{C_2} &\stackrel{\text{def}}{=} 2^{n-1} - |N|_2 = [(2^{n-1} - 1) + 1] - |N|_2 = \\ &= \underbrace{(1 \ 1 \dots 1 \ 1)}_{(n-1) \text{ biți}}_2 - \underbrace{(a_{n-2} a_{n-3} \dots a_1 a_0)}_{(n-1) \text{ biți}}_2 + \underbrace{(0 \ 0 \dots 0 \ 1)}_{(n-1) \text{ biți}}_2 = \\ &= [(1 - a_{n-2}), (1 - a_{n-3}), \dots, (1 - a_1), (1 - a_0)]_2 + (0 \ 0 \dots 0 \ 1)_2 = \\ &= \underbrace{(\bar{a}_{n-2} \bar{a}_{n-3} \dots \bar{a}_1 \bar{a}_0)}_2 + \underbrace{(0 \ 0 \dots 0 \ 1)}_2 \stackrel{\text{not.}}{=} \underbrace{(\tilde{a}_{n-2} \tilde{a}_{n-3} \dots \tilde{a}_1 \tilde{a}_0)}_2; \\ &\quad |N|_{C_1} \quad + \quad 1 \quad = \quad |N|_{C_2} \end{aligned}$$

(suma nu se mai face întotdeauna fără transport și trebuie considerată valoarea de ansamblu în loc de fiecare bit în parte).

– Cunoscând reprezentarea internă, valoarea zecimală externă se determină prin formula de definiție, cu sau fără recomplementare, în funcție de bitul de semn:

$$N = \begin{cases} -(|N|_{C_2})_{10}; \text{ pt. } b_{n-1} = 1 \\ + |N|_{10}; \text{ pt. } b_{n-1} = 0 \end{cases} \Rightarrow N = \begin{cases} - [\boxed{1} \cdot 2^{n-1} - |N|_{10}] \\ - [\boxed{0} \cdot 2^{n-1} - |N|_{10}] \end{cases} \Rightarrow$$

(red arrows point from boxed 1 and 0 to pt. $b_{n-1} = 1$ and pt. $b_{n-1} = 0$ respectively)

$$\begin{aligned}
&\Rightarrow N = -b_{n-1} \cdot 2^{n-1} + \sum_{k=0}^{n-2} b_k \cdot 2^k \Rightarrow \text{metoda (I.)} \\
&\Rightarrow N = \begin{cases} + \sum_{k=0}^{n-2} b_k \cdot 2^k; \text{ pt. } b_{n-1} = 0; \\ - \left[1 \cdot (1 + \sum_{k=0}^{n-2} 2^k) - \sum_{k=0}^{n-2} b_k \cdot 2^k \right]; \text{ pt. } b_{n-1} = 1 \end{cases} \\
&= \begin{cases} + \sum_{k=0}^{n-2} b_k \cdot 2^k \\ - \left[1 + \sum_{k=0}^{n-2} (1 - b_k) \cdot 2^k \right] \end{cases} = \begin{cases} + \sum_{k=0}^{n-2} b_k \cdot 2^k; \text{ pt. } b_{n-1} = 0 \\ - \left[1 + \sum_{k=0}^{n-2} \bar{b}_k \cdot 2^k \right]; \text{ pt. } b_{n-1} = 1 \end{cases} \Rightarrow \\
&\hspace{25em} \bar{b}_{n-1} = 0; \\
&\Rightarrow N = \begin{cases} + [b_{n-1} \cdot 2^{n-1} + \sum_{k=0}^{n-2} b_k \cdot 2^k] \\ - [1 + \bar{b}_{n-1} \cdot 2^{n-1} + \sum_{k=0}^{n-2} \bar{b}_k \cdot 2^k] \end{cases} = \begin{cases} + \sum_{k=0}^{n-1} b_k \cdot 2^k; \text{ pt. } b_{n-1} = 0; \\ - [1 + \sum_{k=0}^{n-1} \bar{b}_k \cdot 2^k]; \text{ pt. } b_{n-1} = 1; \end{cases} \Rightarrow \\
&\Rightarrow N = \begin{cases} + \sum_{k=0}^{n-1} b_k \cdot 2^k; \text{ pt. } b_{n-1} = 0; \\ - \sum_{k=0}^{n-1} \bar{b}_k \cdot 2^k; \text{ pt. } b_{n-1} = 1; \end{cases} \rightarrow \text{metoda (II.)}
\end{aligned}$$

Observații: – Se poate include la recomplementare față de 2 și bitul de semn cu ponderea 2^{n-1} (atât la numere pozitive cât și la numere negative);

– Bitul cu rangul $(n-1)$ contribuie cu o pondere negativă (-2^{n-1}) la valoarea numărului, spre deosebire de cazul numerelor fără semn când formula era *similară*, dar contribuția era pozitivă.

Domeniul de reprezentare se determină cu formula valorii zecimale externe:

$$\begin{aligned}
&\left. \begin{aligned} N_{min} &= -[1] \cdot 2^{n-1} + \sum_{k=0}^{n-2} [0] \cdot 2^k = -2^{n-1} (!) \\ N_{max} &= -[0] \cdot 2^{n-1} + \sum_{k=0}^{n-2} [1] \cdot 2^k = +(2^{n-1} - 1) \end{aligned} \right\} \Rightarrow -2^{n-1} \leq N \leq 2^{n-1} - 1; \\
&\hspace{15em} \downarrow \\
&\hspace{15em} \text{Există posibilitatea de a} \\
&\hspace{15em} \text{reprezenta un număr negativ în} \\
&\hspace{15em} \text{plus față de numerele pozitive.}
\end{aligned}$$

Dacă: $n = 8 \Rightarrow -128 \leq N \leq +127 \Rightarrow$ sunt **128** de numere *negative* și **127** de numere *pozitive*.

Exemplu: $n = 6$ biți; $N = -27 \xrightarrow{C_2} ?$

$$|N|_2 = \left| \overline{11011} \right| = 11011 \Rightarrow |N|_{C_2} = |N|_{C_1} + 1 =$$

$\underbrace{\hspace{1.5cm}}_{b_5=1}$

$$= (\overline{11011})_2 + (00001)_2 =$$

$$= (00100)_2 + (00001)_2 = (00101)_2 \Rightarrow$$

$$\Rightarrow N_{C_2} = \boxed{100101};$$

$$N \text{ cu formula} = -b_5 \cdot 2^5 + \sum_{k=0}^4 b_k \cdot 2^k =$$

$$= -1 \cdot 32 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$$

$$= -32 + 4 + 1 = \boxed{-27};$$

$$N \text{ recomplementare} = -[(\overline{100101})_2 + (000001)_2] =$$

$$= -[(011010)_2 + (000001)_2] = -(011011)_2 = \boxed{(-27)_{10}}.$$

Observații: – În reprezentarea C_2 există un singur cod pentru zero, aceasta fiind și explicația pentru faptul că “încapă” și codul pentru (-2^{n-1}) la reprezentare:

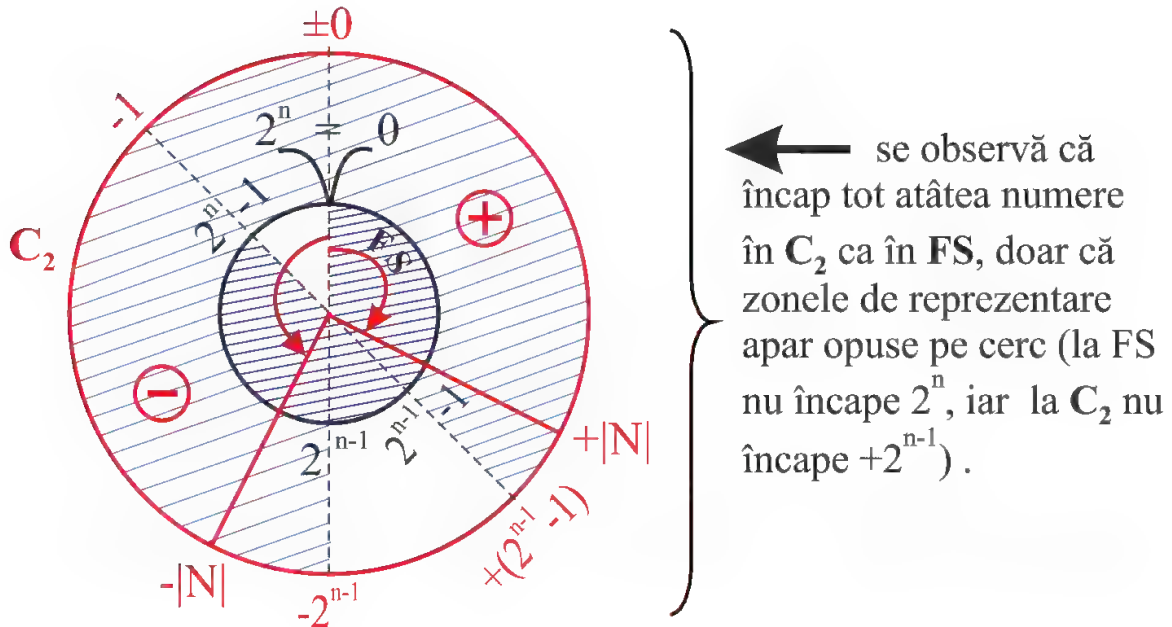
$$0 = \begin{cases} +0 \xrightarrow{C_2} \overline{000\dots00}; \\ \quad \quad \quad (n-1) \text{ biți} \\ -0 \xrightarrow{C_2} \overline{100\dots00} = 1[(\overline{11\dots11})_2 + (\overline{00\dots01})_2] = \\ \quad \quad \quad (n-1) \text{ biți} \quad (n-1) \text{ biți} \quad (n-1) \text{ biți} \end{cases}$$

$$= [(\underbrace{1+1}_{\text{poziția } b_{n-1}}), \underbrace{00\dots00}_{(n-1) \text{ biți}}]_2 = (\overline{000\dots00})_2 = \underbrace{000\dots00}_{n \text{ biți}} = +0$$

$\underbrace{\hspace{1.5cm}}_{\text{poziția } b_{n-1}}$

Depășește reprezentarea pe $\boxed{n}$ biți modulo 2^n și SE IGNORĂ

– La reprezentarea pe cercul modulo 2^n a codurilor FS (fără semn) și C_2 (cod complementar față de 2), se observă câștigarea sectorului dintre $(2^n - 1)$ și (2^n) , care se pierdea în C_1 datorită codului pentru “-0”:



Reguli practice pentru coduri complementare:

– Metode de
obținere a
reprezentării
 C_2 :

Se inversează toți biții numărului și se adună 1 la poziția cea mai puțin semnificativă, neglijându-se transportul ce poate apărea din poziția $(n-2)$ spre rangul semnului $(n-1)$:

$$\begin{aligned} b_{n-2}b_{n-3} \dots b_1b_0 &= \bar{a}_{n-2}\bar{a}_{n-3} \dots \bar{a}_1\bar{a}_0 = \\ &= \bar{a}_{n-2}\bar{a}_{n-3} \dots \bar{a}_1\bar{a}_0 + 00 \dots 01; \end{aligned}$$

Se inversează toți biții numărului, dar începând după primul 1, cu sensul de parcurgere de la rangul (0) spre rangul $(n-2)$:

$$\begin{aligned} b_{n-2}b_{n-3} \dots b_1b_0 &= \bar{a}_{n-2}\bar{a}_{n-3} \dots \bar{a}_1\bar{a}_0 = \\ &= \bar{a}_{n-2}\bar{a}_{n-3} \dots \bar{a}_{k+1} \underbrace{1}_{\text{primul 1 în poziția k}} a_k \dots a_1a_0; \end{aligned}$$

Biți inversați după
primul 1

Biți neinversați
care erau 0

primul 1 în
poziția k

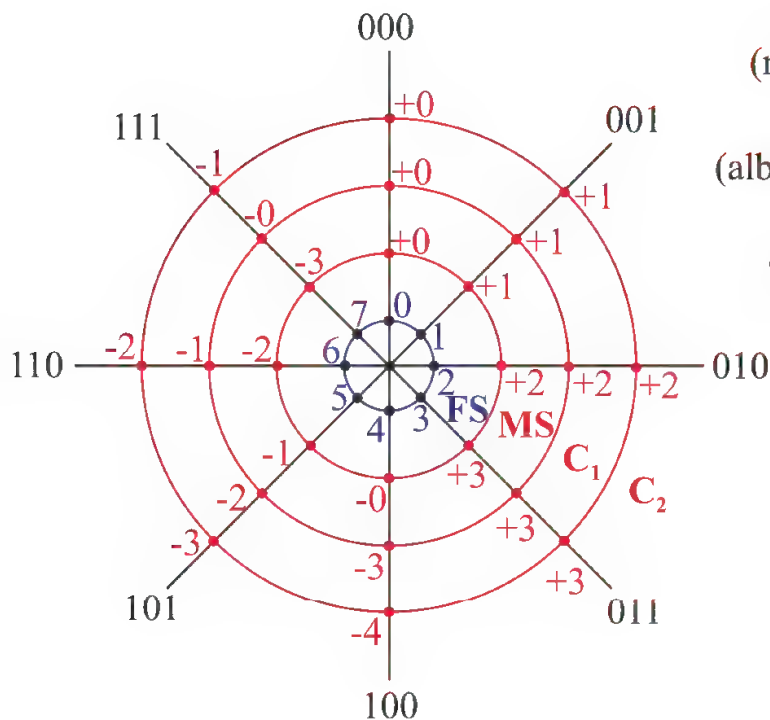
(algoritmul apare din propagarea transportului
la adunarea cu 1 în baza doi)

– Metode de obținere a valorii zecimale reprezentate prin C_1 sau C_2 :

dacă: $b_{n-1} = 0$, se substituie bitul de semn cu semnul “+”;

dacă: $b_{n-1} = 1$, se recomplementează (față de 1 sau 2) toți biții, inclusiv bitul de semn și se atașează “-” în față.

– Comparație între semnificațiile codurilor binare interne pe 3 biți, interpretate ca valoare FS, MS, C_1 și C_2 :



(negru) “•”= valoare binară internă pe 3 biți;

(albastru) “•”= interpretare externă fără semn (FS);

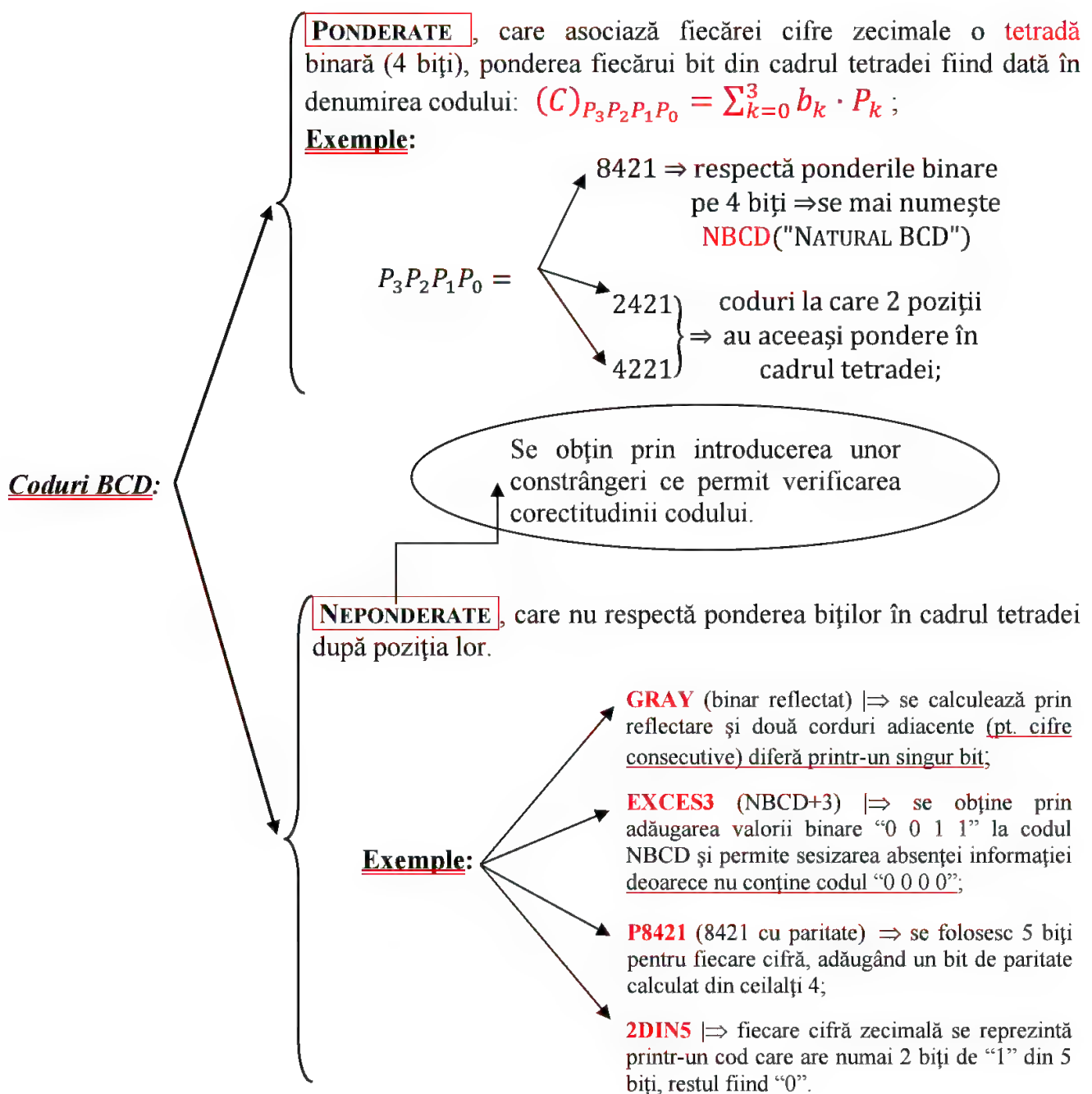
(roșu) “•”= interpretări externe cu semn în MS, C_1 și C_2 ;

– Semicercul negativ C_1 este opus față de MS, iar la C_2 este rotit cu un pas în sensul acelor de ceasornic.

3.3. Reprezentarea directă a cifrelor zecimale

Valorile numerice zecimale externe se pot reprezenta direct în format intern prin codificarea cifrelor zecimale în mod independent (nu valoarea lor de ansamblu, ci fiecare separat), folosind câte 4 biți pentru fiecare cifră $\Rightarrow$

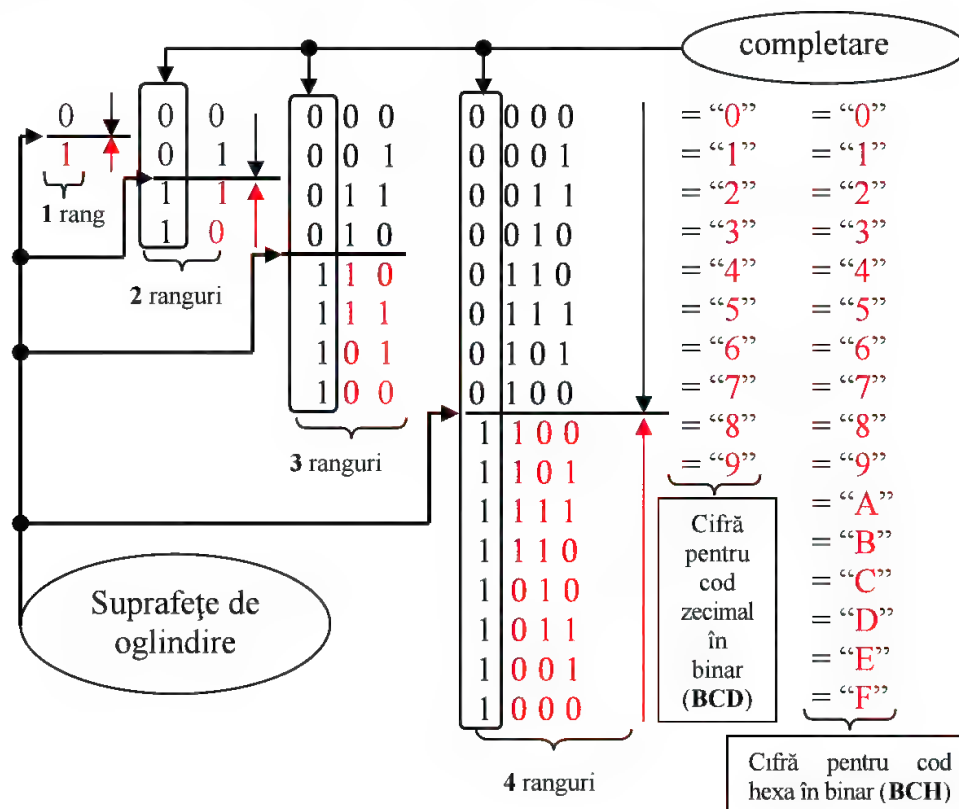
$\Rightarrow$ reprezentare în cod **zecimal-binar** sau **BCD** = "BINARY CODED DECIMAL".



Codul GRAY: – Este neponderat, deci valoarea cifrei zecimale nu se poate determina prin sumarea ponderată a biților;

– Trecerea de la o cifră zecimală la următoarea se face prin modificarea unui singur bit (proprietate de adiacență), fiind util pentru mărimile care cresc succesiv în pași (traductoare unghiulare de poziție sau diagrame de minimizare a funcțiilor logice).

– Construcția codului se face în mod succesiv pentru fiecare rang, prin oglundirea codului cu mai puține ranguri, adăugând pe verticală biți “0” deasupra suprafeței de oglindire și biți “1” sub această suprafață, înaintea codului vechi oglindit:



Cifra zecimală	PONDERATE			NEPONDERATE		
	8 4 2 1	2 4 2 1	4 2 2 1	EXCES 3	GRAY	2 DIN 5
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 1 1
1	0 0 0 1	0 0 0 1	0 0 0 1	0 1 0 0	0 0 0 1	0 0 1 0 1
2	0 0 1 0	0 0 1 0	0 0 1 0	0 1 0 1	0 0 1 1	0 0 1 1 0
3	0 0 1 1	0 0 1 1	0 0 1 1	0 1 1 0	0 0 1 0	0 1 0 0 1
4	0 1 0 0	0 1 0 0	0 1 1 0	0 1 1 1	0 1 1 0	0 1 0 1 0
5	0 1 0 1	1 0 1 1	1 0 0 1	1 0 0 0	0 1 1 1	0 1 1 0 0
6	0 1 1 0	1 1 0 0	1 1 0 0	1 0 0 1	0 1 0 1	1 0 0 0 1
7	0 1 1 1	1 1 0 1	1 1 0 1	1 0 1 0	0 1 0 0	1 0 0 1 0
8	1 0 0 0	1 1 1 0	1 1 1 0	1 0 1 1	1 1 0 0	1 0 1 0 0
9	1 0 0 1	1 1 1 1	1 1 1 1	1 1 0 0	1 1 0 1	1 1 0 0 0

Cele două seturi de cifre zecimale
complementare față de 9 (adică $0 \div 4$ și $5 \div 9$)
au codurile obținute prin inversarea biților.

3.4. Reprezentarea în virgulă mobilă a numerelor reale

– Se utilizează la reprezentarea numerelor reale supra-unitare cu valoare foarte mare sau sub-unitare cu valoare foarte mică, care nu pot fi reprezentate în formatul cu virgulă fixă folosind un număr rezonabil de biți.

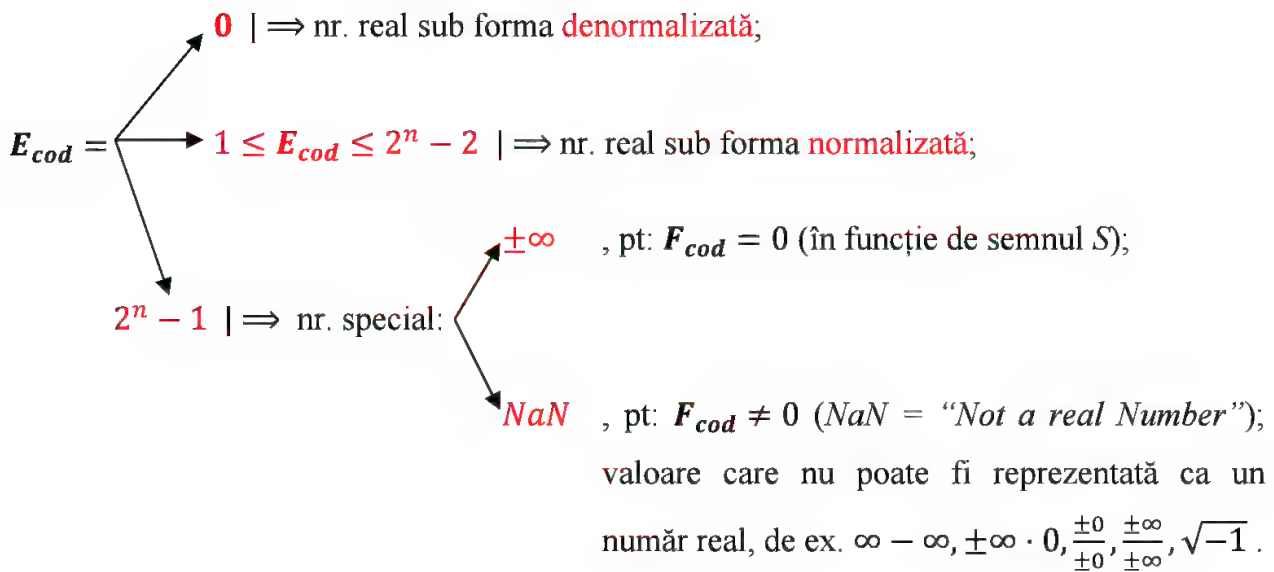
– Numerele se scriu în **formatul științific**: $(-1)^S \cdot F \cdot B^E$, ca un produs cu semnul dat de S între o fracție cu virgulă fixă F (mantisă) și un exponent E al bazei B utilizate; prin modificarea valorii exponentului, se schimbă practic poziția virgulei în cadrul produsului final, chiar dacă fracția are virgula fixată implicit după prima cifră.

– În sistemele de calcul baza codificării este $B = 2$, iar din fracția F se reprezintă doar cifrele de după virgulă F_{cod} ; prima cifră urmată de virgula fixată implicit nu se mai reprezintă, pentru a se economisi biți:

– Dacă

- ↗ codificarea pentru exponent este nulă, atunci și prima cifră a fracției este considerată “0”: $E_{cod} = 0 \mid \Rightarrow F = 0.F_{cod}$ (fracție **denormalizată**);
- ↘ codificarea exponentului este diferită de zero, atunci prima cifră a fracției fixe este considerată “1”: $E_{cod} \neq 0 \mid \Rightarrow F = 1.F_{cod}$ (fracție **normalizată**);

– Din gama de valori posibile pentru codificarea exponentului pe n biți ($0 \dots 2^n - 1$), se rezervă câte o valoare în ambele capete ale intervalului pentru cazurile speciale:



– Pentru a putea reprezenta atât valori supra-unitare cât și valori sub-unitare, exponentul trebuie să aibă semnificație atât pozitivă cât și negativă; nu se recurge însă la complementul față de 2 cu bit de semn, ci la o codificare mai simplă de tip **EXCES negativ** cu minus jumătatea domeniului de reprezentare al formei normalizate: $\frac{(2^n-2)}{2} = 2^{n-1} - 1$.

– Practic, interpretarea biților din reprezentarea exponentului nu este directă, ci valoarea lui reală se obține din cea codificată printr-o scădere implicită, care se sub-înțelege:

$$E = E_{cod} - (2^{n-1} - 1)$$

Observații: – Exponentul nul care marchează reprezentarea sub forma denormalizată este interpretat tot prin același exces negativ, dar **considerând codificarea minimă posibilă din forma normalizată**, adică ca și cum $E_{cod} = 0 + 1$:

$$E_{cod} = 0 \Rightarrow E = (0 + 1) - (2^{n-1} - 1) \Rightarrow E = 0 - (2^{n-1} - 2)$$

– Deoarece în forma denormalizată primul bit al fracției este “0” implicit și exponentul se limitează la echivalentul cel mai mic din forma normalizată, se pot reprezenta astfel **numere mai apropiate de zero** (de o parte și de cealaltă a axei reale) decât cele în forma normalizată;

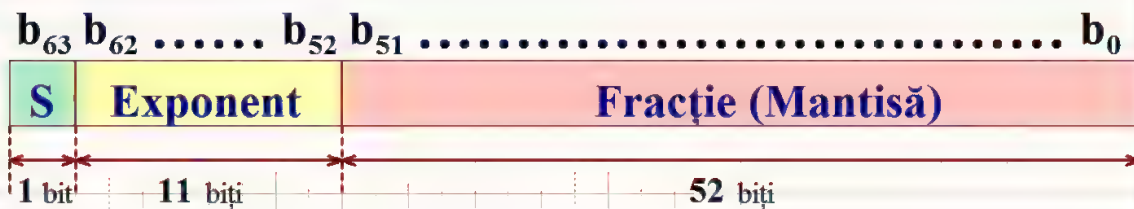
– Reprezentarea **numărului real zero** este posibilă tot numai în forma denormalizată (cu $F = 0, F_{cod} = 0.00 \dots 0$), din cauză că forma normalizată are primul bit al fracției “1” implicit; există două codificări distincte pentru zero, în funcție de semn.

– Standardul internațional **IEEE 754** introduce 3 variante de reprezentare în virgulă mobilă, în funcție de numărul total de biți utilizați: **simplă precizie** (pe 32 de biți), **dublă precizie** (pe 64 de biți) și **precizie extinsă** (pe 80 de biți), ultima variantă fiind mai puțin folosită în prezent.

Simplă precizie (IEEE 754 pe 32 de biți):



Dublă precizie (IEEE 754 pe 64 de biți):



– Valoarea reprezentată:

- cu **simplă precizie**
 - normalizată: $(-1)^S \cdot [1.F]_2 \cdot 2^{E-127}$;
 - denormalizată: $(-1)^S \cdot [0.F]_2 \cdot 2^{0-126}$;
- ($E = \overline{1 \dots 254}$; pt. $E = 255 \Rightarrow \pm\infty, NaN$)
- cu **dublă precizie**
 - normalizată: $(-1)^S \cdot [1.F]_2 \cdot 2^{E-1023}$;
 - denormalizată: $(-1)^S \cdot [0.F]_2 \cdot 2^{0-1022}$;
- ($E = \overline{1 \dots 2046}$; pt. $E = 2047 \Rightarrow \pm\infty, NaN$)

Exemple: – Reprezentarea lui π cu virgulă mobilă în **simplă precizie**:

$$\begin{aligned}\pi &= 0100\ 0000\ 0100\ 1001\ 0000\ 1111\ 1101\ 1011_2 \\ &= 4049\ 0FDB\ H = +[1.F]_2 \cdot 2^{128-127} = 13176795 \cdot 2^{1-23} \\ &\cong 3.1415927410125732 \quad (\text{aproximare cu 6 zecimale exacte});\end{aligned}$$

– Reprezentarea lui π cu virgulă mobilă în **dublă precizie**:

$$\begin{aligned}\pi &= 0100\ 0000\ 0000\ 1001\ 0010\ 0001\ 1111\ 1011\ 0101\ 0100\ 0100\ 0100\ 0010\ 1101\ 0001\ 1000_2 \\ &= 4009\ 21FB\ 5444\ 2D18\ H = +[1.F]_2 \cdot 2^{1024-1023} = 7074237752028440 \cdot 2^{1-52} \\ &\cong 3.1415926535897931 \quad (\text{aproximare cu 15 zecimale exacte}).\end{aligned}$$

– Domeniul de reprezentare este simetric pentru numerele negative și cele pozitive:

Precizie	Forma	Valoarea minimă pozitivă	Valoarea maximă pozitivă
Simplă	Normalizată:	$(S = 0, E = 1, F = 00...00_2)$ $N_{min} = \mathbf{0080\ 0000\ H}$ $= +[1.\mathbf{00\ ...00}]_2 \cdot 2^{1-127}$ $= 1 \cdot 2^{-126}$ $\cong 1.17549435 \cdot 10^{-38}$	$(S = 0, E = 254, F = 11...11_2)$ $N_{max} = \mathbf{7F7F\ FFFF\ H}$ $= +[1.\mathbf{11\ ...11}]_2 \cdot 2^{254-127}$ $= (2 - 2^{-23}) \cdot 2^{127}$ $\cong 3.40282347 \cdot 10^{38}$
	Denormalizată:	$(S = 0, E = 0, F = 00...01_2)$ $D_{min} = \mathbf{0000\ 0001\ H}$ $= +[0.\mathbf{00\ ...01}]_2 \cdot 2^{0-126}$ $= 2^{-23} \cdot 2^{-126}$ $\cong 1.40129846 \cdot 10^{-45}$	$(S = 0, E = 0, F = 11...11_2)$ $D_{max} = \mathbf{007F\ FFFF\ H}$ $= +[0.\mathbf{11\ ...11}]_2 \cdot 2^{0-126}$ $= (1 - 2^{-23}) \cdot 2^{-126}$ $\cong 1.17549421 \cdot 10^{-38} < N_{min}$ (diferă după 6 cifre zecimale)
Dublă	Normalizată:	$(S = 0, E = 1, F = 00...00_2)$ $N_{min} =$ $= \mathbf{0010\ 0000\ 0000\ 0000\ H}$ $= +[1.\mathbf{00\ ...00}]_2 \cdot 2^{1-1023}$ $= 1 \cdot 2^{-1022}$ $\cong 2.22507386 \cdot 10^{-308}$	$(S = 0, E = 2046, F = 11...11_2)$ $N_{max} =$ $= \mathbf{7FEF\ FFFF\ FFFF\ FFFF\ H}$ $= +[1.\mathbf{11\ ...11}]_2 \cdot 2^{2046-1023}$ $= (2 - 2^{-52}) \cdot 2^{1023}$ $\cong 1.79769313 \cdot 10^{308}$
	Denormalizată:	$(S = 0, E = 0, F = 00...01_2)$ $D_{min} =$ $= \mathbf{0000\ 0000\ 0000\ 0001\ H}$ $= +[0.\mathbf{00\ ...01}]_2 \cdot 2^{0-1022}$ $= 2^{-52} \cdot 2^{-1022} =$ $\cong 4.94065646 \cdot 10^{-324}$	$(S = 0, E = 0, F = 11...11_2)$ $D_{max} =$ $= \mathbf{000F\ FFFF\ FFFF\ FFFF\ H}$ $= +[0.\mathbf{11\ ...11}]_2 \cdot 2^{0-1022}$ $= (1 - 2^{-52}) \cdot 2^{-1022}$ $\cong 2.22507386 \cdot 10^{-308} < N_{min}$ (diferă după 14 cifre zecimale)

Observație: – În dublă precizie nu este importantă atât lărgirea domeniului de reprezentare de la numere mult mai mici la numere mult mai mari decât în simplă precizie, cât mai ales **posibilitatea reprezentării mult mai fine a valorilor reale din domeniul obișnuit**, datorată creșterii numărului de combinații binare mai mult pentru fracție decât pentru exponent !

3.5. Reprezentarea caracterelor alfa-numerice

– Se folosesc pentru reprezentarea internă a informațiilor de tip text, cu simboluri grafice formate din cifrele de la zero la nouă, literele alfabetului, semnele matematice și de punctuație, denumite în general **caractere** (desenele grafice ale simbolurilor respective).

– Pentru reprezentarea acestor caractere în format intern este nevoie de minim

7 biți $\Rightarrow$ cu care se pot codifica ($2^7 = 128$) de combinații distincte.

– Cel mai folosit cod cu acceptare internațională este

Codul ASCII = “AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE” :

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

– Codurile {**00...1F, 7F**} sunt folosite pentru caractere speciale **de control** care nu se pot tipări sau afișa, utilizate în trecut la controlul transmisiei datelor și al imprimării, depășite în prezent, cu excepția: **HT** = “Horizontal Tab” (aliniere orizontală), **LF** = “Line Feed” (linie nouă) și **CR** = “Carriage Return” (coloană nouă);

– Codurile {**20...2F, 3A...40, 5B...60, 7B...7E**} sunt ale caracterelor folosite pentru reprezentarea **semnelor matematice și de punctuație**;

– Codurile {**30...39**} sunt ale caracterele pentru **cifrele numerice** zecimale de la zero la nouă;

– Codurile {**41...5A**} sunt pentru **caracterele literelor mari** ale alfabetului;

– Codurile {**61...7A**} sunt cele pentru **caracterele literelor mici** ale alfabetului.

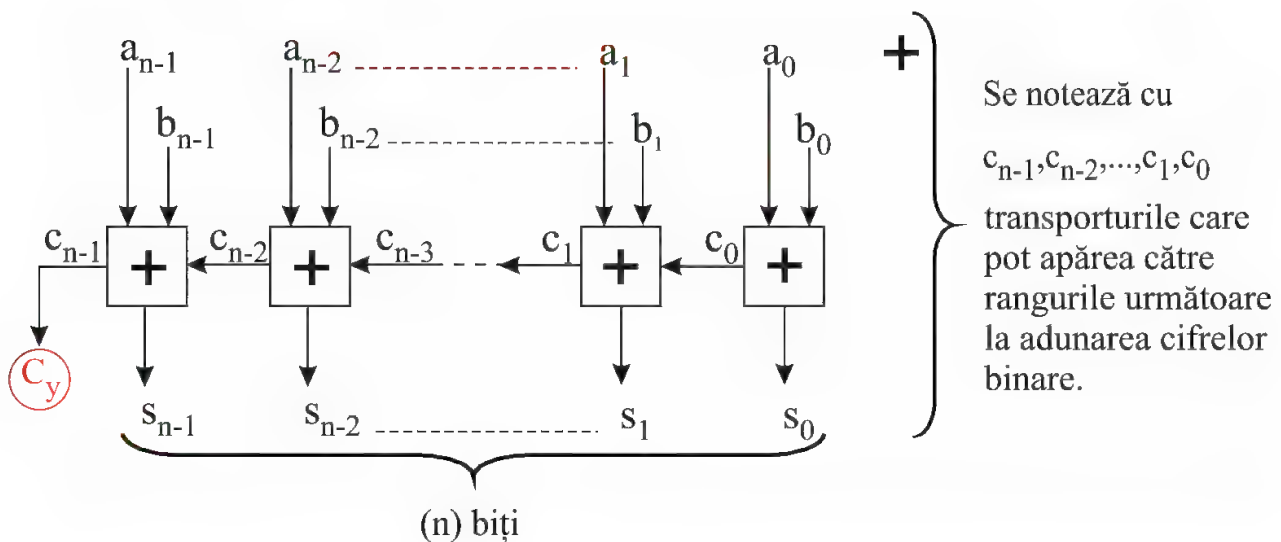
4. Operații aritmetice binare

4.1. Operații cu numere fără semn (FS)

– Se consideră două numere cu valorile zecimale externe **A** și **B**, reprezentate în calculator folosind **n** biți fără semn:

$$\begin{cases} A = (a_{n-1}a_{n-2} \dots a_1a_0)_2; 0 \leq A \leq 2^n - 1; \\ B = (b_{n-1}b_{n-2} \dots b_1b_0)_2; 0 \leq B \leq 2^n - 1; \end{cases}$$

– Adunarea: **A + B = S**;

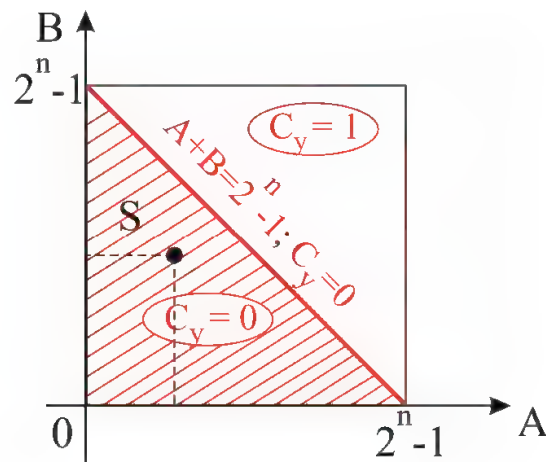


– Dacă există transport din rangul (n-1), suma a depășit domeniul de reprezentare pe (n) biți și ar fi necesar un bit suplimentar care se notează:

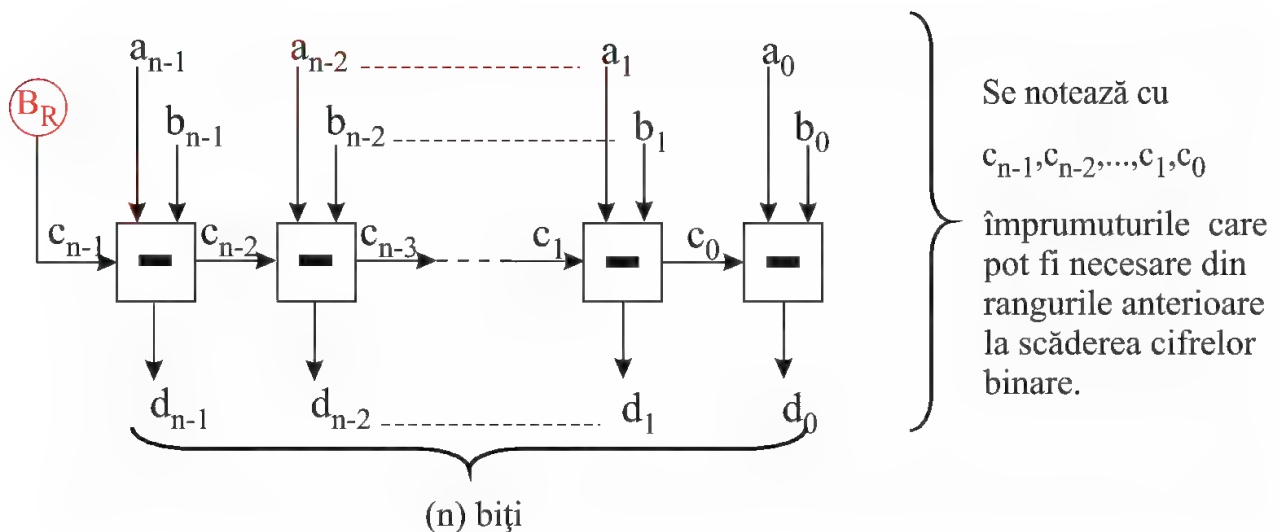
Cy = "CARRY" = indicator de depășire; rezultatul $S = (s_{n-1}s_{n-2} \dots s_1s_0)_2$ este suma modulo (2^n) pe (n) biți.

$$C_y = \begin{cases} 0; \text{pt. } A+B=S \leq 2^n-1; \\ 1; \text{pt. } A+B=S > 2^n-1 \Leftrightarrow A+B=S \geq 2^n; \end{cases} C_y = c_{n-1};$$

– Domeniul de valori externe pentru care nu apare depășire la adunare este când suma S se află în interiorul sau pe granița triunghiului hașurat:



– Scăderea: $A - B = D$;

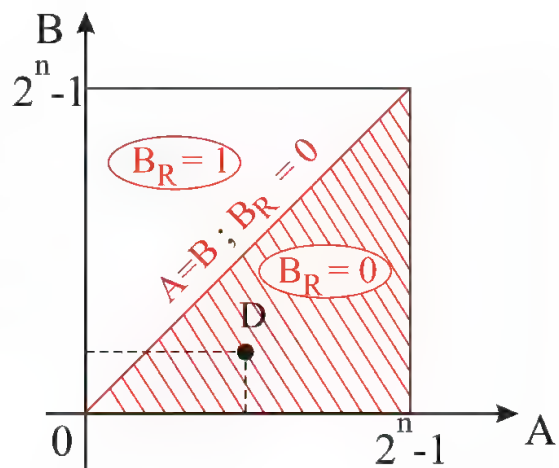


– Dacă există împrumut spre rangul $(n-1)$, înseamnă că $A < B$ și diferența a depășit domeniul de reprezentare, fiind negativă și nu se poate realiza decât cu un împrumut multiplu din toți biții fictivi superiori, care vor fi lăsați cu valori “1” după împrumut;

– Considerând doar primul bit fictiv de rang (n) , acesta poate fi folosit ca indicator de depășire: $B_R = \text{“BORROW”}$;

$$B_R = \begin{cases} 0; \text{ pt. } A-B=D \geq 0 \Leftrightarrow A \geq B ; \\ 1; \text{ pt. } A-B=D < 0 \Leftrightarrow A < B ; \end{cases} B_R = c_{n-1} ;$$

– Domeniul de valori externe pentru care nu apare depășire la scădere este când diferența **D** se află în interiorul sau pe laturile triunghiului hașurat:

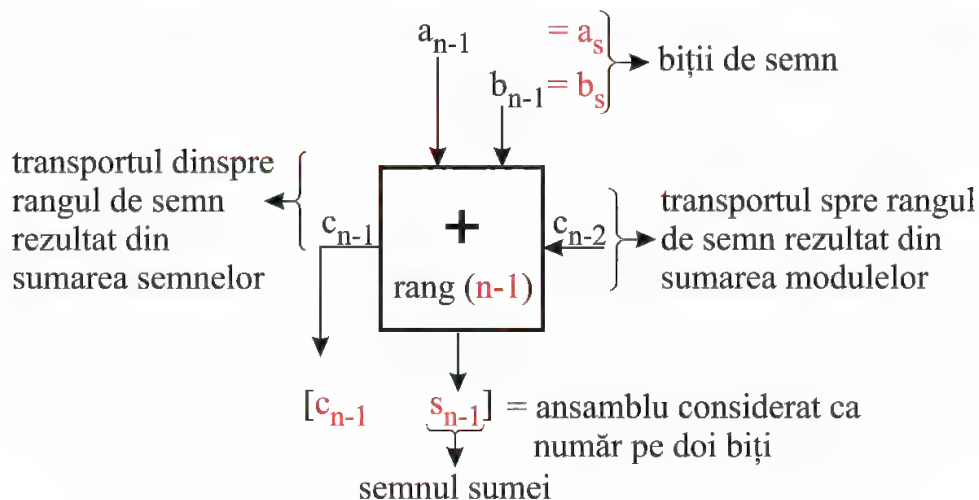


Observație: – Deoarece indicatorii C_y și B_R au același rol, dar în cazul unor operații diferite, se poate folosi un singur bit indicator de depășire, atât la adunare cât și la scădere, notat C_y = transport între rangul (n) și rangul ($n-1$):

$$\text{rang}(n) \xrightarrow[\text{adunare}]{\text{scădere } C_y} \text{rang}(n-1)$$

4.2. Operații între numere cu semn în C_2

- Operațiile cu reprezentări interne în C_1 necesită etape suplimentare de corecție (datorită codului pentru “-0”) și nu se folosesc la implementarea în hardware;
- Se lucrează exclusiv cu reprezentări în C_2 , iar biții de semn se sumează obișnuit, folosind aceeași structură fizică de la numere fără semn.
- Detectia depășirilor se face considerând transportul **spre** și cel **dinspre** rangul de semn:



– Pentru analiza apariției depășirilor, se calculează valoarea internă echivalentă a reprezentărilor în cod C_2 :

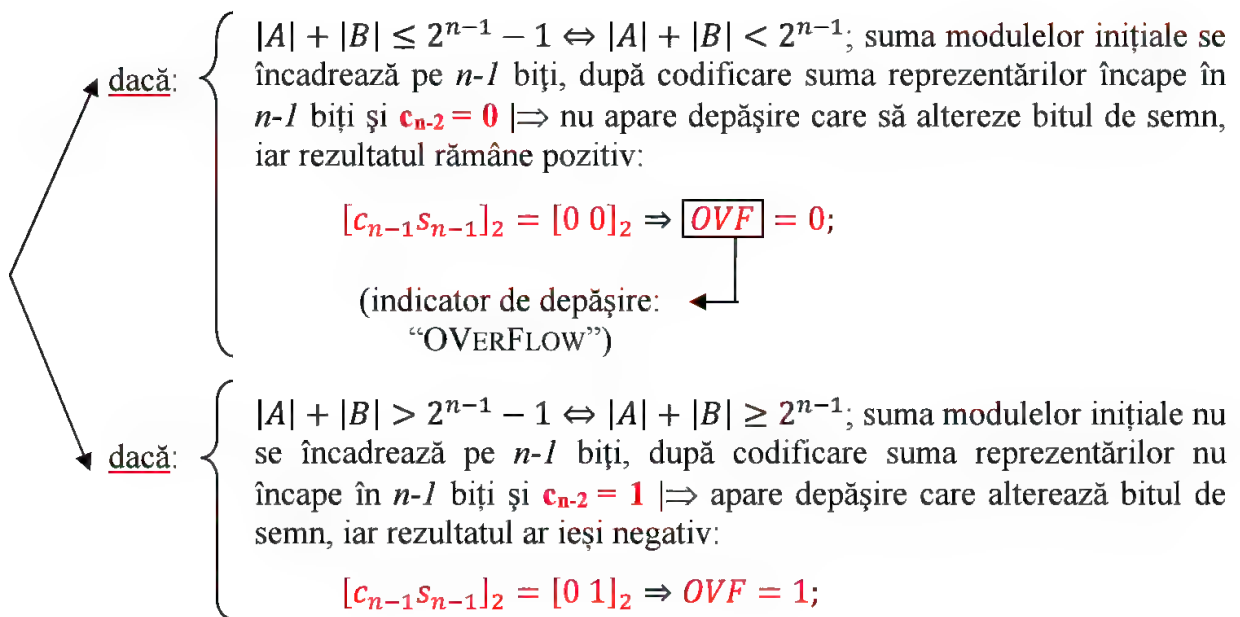
$$N = \begin{cases} +|N|_2 \xrightarrow{c_2} [0|N|_2] = |N|_2 \\ -|N|_2 \xrightarrow{c_2} [1|N|_2] = 2^{n-1} + |N|_{C_2} = 2^{n-1} + \overbrace{(|N|_{C_2})}^{2^{n-1} - |N|_2} \end{cases} \Rightarrow$$

$$\Rightarrow N_{C_2} = \begin{cases} |N|; \text{ pt. } 0 \leq N \leq 2^{n-1}-1; \\ 2^n - |N|; \text{ pt. } -2^{n-1} \leq N \leq 0; \end{cases}$$

[la numerele negative, datorită bitului de semn “1”, în calculator apare valoarea complementată față de doi pe n biți a modulului, conform definiției.]

4.2.1. Numere pozitive: $0 \leq A, B \leq 2^{n-1} - 1$;

$$\begin{matrix} a_s = 0 \\ b_s = 0 \end{matrix} \Rightarrow \begin{cases} [c_{n-1}s_{n-1}]_2 = 0 + 0 + c_{n-2} = [0 \ c_{n-2}]_2; \\ S_{C_2} = A_{C_2} + B_{C_2} = |A| + |B| \Rightarrow S \geq 0; \end{cases}$$



Exemple: $n = 8$ biți; $A = +120$; $B = +5$;

$$|A|_2 = 1111000 \Rightarrow A_{c_2} = 01111000 +$$

$$|B|_2 = \underbrace{0000101}_{7 \text{ biți}} \Rightarrow B_{c_2} = 0000101$$

$$S_{c_2} = 01111101 \Rightarrow S = +125;$$

$c_{n-1}=0;$ $c_{n-2}=0;$

$n = 8$ biți; $A = +120$; $B = +8$;

$$|A|_2 = 1111000 \Rightarrow A_{c_2} = 01111000 +$$

$$|B|_2 = \underbrace{0001000}_{7 \text{ biți}} \Rightarrow B_{c_2} = 00001000$$

$$S_{c_2} = 10000000 \Rightarrow S = -128;$$

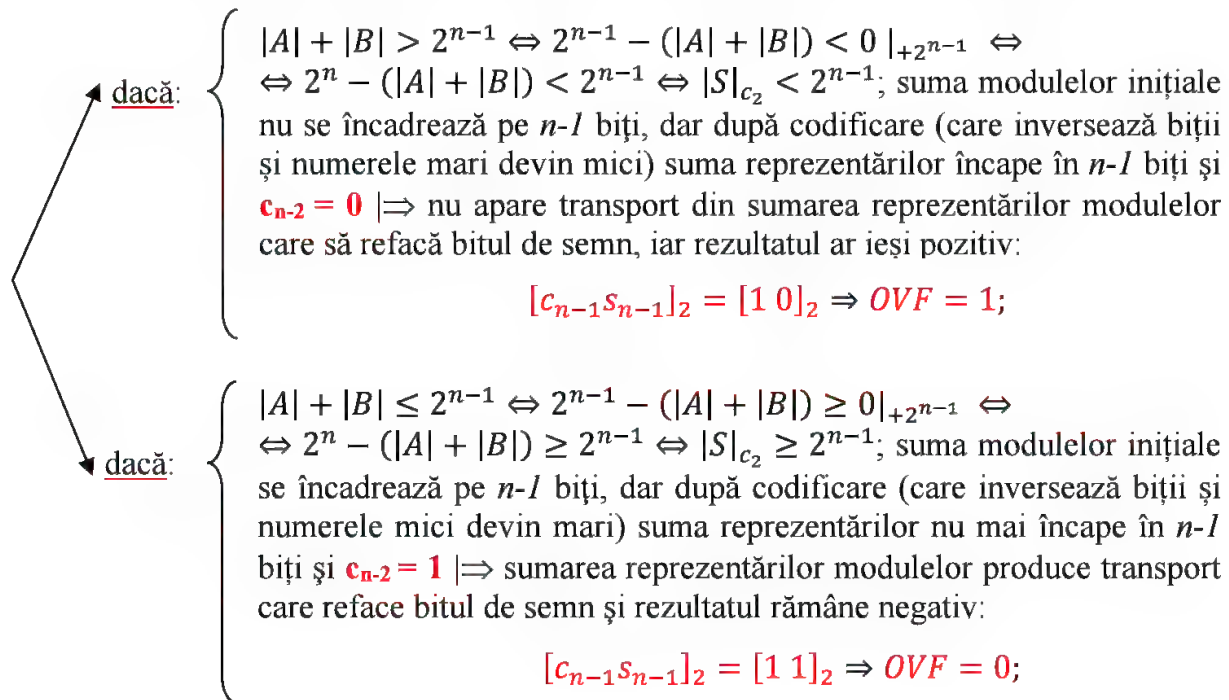
$c_{n-1}=0;$ $c_{n-2}=1;$

$$\text{greșit} \Rightarrow \begin{cases} OVF = 1 \\ S = +128 \end{cases}$$

4.2.2. Numere negative: $-2^{n-1} < A, B < 0$;

$$\begin{cases} a_s = 1 \\ b_s = 1 \end{cases} \Rightarrow \begin{cases} [c_{n-1}s_{n-1}]_2 = 1 + 1 + c_{n-2} = 2 + c_{n-2} = [1c_{n-2}]_2; \\ S_{c_2} = A_{c_2} + B_{c_2} = (2^n - |A|) + (2^n - |B|) = 2^n + [2^n - (|A| + |B|)] = \\ = \boxed{2^n} + \underbrace{(2^n - |S|)}_{|S|_{c_2}} \Rightarrow S < 0; \\ \downarrow \\ \text{transport spre rang (n): } \boxed{c_{n-1} = 1} \end{cases}$$

Observație: – Transportul dinspre rangul de semn apare totdeauna la sumarea numerelor negative datorită sumei biților de semn și el trebuie ignorat în rezultat.

**Exemple:**

n = 8 biți; A = -120; B = -5;

inversare

$$|A|_2 = 1111000 \Rightarrow |A|_{c_2} = 0001000 \Rightarrow A_{c_2} = 10001000 +$$

$$|B|_2 = 0000101 \Rightarrow |B|_{c_2} = 1111011 \Rightarrow B_{c_2} = 11111011$$

7 biți 7 biți

$$S_{c_2} = \cancel{1}10000011 \Rightarrow$$

c_{n-1}=1 c_{n-2}=1 inversare

$$S_2 = -\underbrace{1111101}_{7 \text{ biți}} \Rightarrow \boxed{S = -125};$$

n = 8 biți; A = -120; B = -11;

$$|A|_2 = 1111000 \Rightarrow |A|_{c_2} = 0001000 \Rightarrow A_{c_2} = 10001000 +$$

$$|B|_2 = 0001011 \Rightarrow |B|_{c_2} = 1110101 \Rightarrow B_{c_2} = 11110101$$

7 biți 7 biți

$$S_{c_2} = \cancel{1}0111101 \Rightarrow$$

c_{n-1}=1; c_{n-2}=0;

$$\Rightarrow S_2 = +\underbrace{1111101}_{7 \text{ biți}} \Rightarrow \boxed{S = +125};$$

greșit $\Rightarrow \begin{cases} OVF = 1 \\ S = -131 \end{cases}$

4.2.3. Numere cu semne opuse: $\begin{cases} 0 \leq A \leq 2^{n-1} - 1; \\ -2^{n-1} \leq B < 0; \end{cases}$

$$\begin{aligned} \boxed{\begin{matrix} a_s = 0 \\ b_s = 1 \end{matrix}} &\Rightarrow \begin{cases} [c_{n-1}s_{n-1}]_2 = 0 + 1 + c_{n-2} = \begin{cases} [0\ 1]_2; \text{ pt. } c_{n-2} = 0 \\ [1\ 0]_2; \text{ pt. } c_{n-2} = 1 \end{cases} \Rightarrow \\ \Rightarrow [c_{n-1}s_{n-1}]_2 = [c_{n-2}\overline{c_{n-2}}]_2; \\ S_{c_2} = A_{c_2} + B_{c_2} = |A| + (2^n - |B|) \Rightarrow 2^n + (|A| - |B|) = \\ = \boxed{2^{n-1}} + [2^{n-1} + (|A| - |B|)] \Rightarrow S < 0 \text{ sau } S > 0; \end{cases} \\ &\quad \downarrow \\ &\quad \text{"1" în poziția semnului} \end{aligned}$$

$$\begin{aligned} \text{dacă: } &\begin{cases} |A| < |B| \Leftrightarrow |A| - |B| < 0 \mid_{+2^{n-1}} \Leftrightarrow 2^{n-1} + (|A| - |B|) < 2^{n-1}; \\ \text{numărul pozitiv este mai mic decît cel negativ în modul, diferența} \\ \text{modulelor inițiale consumă prin împrumut bitul } 2^{n-1} \text{ și se încadrează} \\ \text{pe } n-1 \text{ biți, după codificare suma reprezentărilor încapă și ea pe } n-1 \text{ biți} \\ \text{și } c_{n-2} = 0 \mid \Rightarrow \text{sumarea reprezentărilor modulelor nu generează transport} \\ \text{și rămâne "1" în poziția semnului, deci suma este negativă:} \\ c_{n-2} = 0 \Rightarrow [c_{n-1}s_{n-1}]_2 = [0\ 1]_2 \Rightarrow OVF = 0; \end{cases} \\ \text{dacă: } &\begin{cases} |A| \geq |B| \Leftrightarrow |A| - |B| \geq 0 \mid_{+2^{n-1}} \Leftrightarrow 2^{n-1} + (|A| - |B|) \geq 2^{n-1}; \\ \text{numărul pozitiv este mai mare sau egal cu cel negativ în modul,} \\ \text{diferența modulelor inițiale nu consumă prin împrumut bitul } 2^{n-1} \text{ și nu} \\ \text{se încadrează pe } n-1 \text{ biți, după codificare suma reprezentărilor nu} \\ \text{încapă nici ea pe } n-1 \text{ biți și } c_{n-2} = 1 \mid \Rightarrow \text{apare transport spre rangul} \\ \text{semnului prin sumarea reprezentărilor modulelor, care se adună cu bitul} \\ \text{"1" deja existent și îl anulează, deci suma rezultă pozitivă:} \\ c_{n-2} = 1 \Rightarrow [c_{n-1}s_{n-1}]_2 = [1\ 0]_2 \Rightarrow OVF = 0; \end{cases} \end{aligned}$$

Exemple: $n = 8$ biți; $A = +78$; $B = -120$;

$$\begin{aligned} |A|_2 &= 1\ 0\ 0\ 1\ 1\ 1\ 0 \Rightarrow |A|_{c_2} = 1\ 0\ 0\ 1\ 1\ 1\ 0 \Rightarrow A_{c_2} = \underline{0\ 1\ 0\ 0\ 1\ 1\ 1\ 0} + \\ |B|_2 &= \underline{1\ 1\ 1\ 1\ 0\ 0\ 0} \Rightarrow |B|_{c_2} = \underline{0\ 0\ 0\ 1\ 0\ 0\ 0} \Rightarrow B_{c_2} = \underline{1\ 0\ 0\ 0\ 1\ 0\ 0\ 0} \\ &\quad \text{7 biți} \qquad \qquad \text{7 biți} \\ \Rightarrow S_2 &= -\underline{0\ 1\ 0\ 1\ 0\ 1\ 0} \Rightarrow \boxed{S = -42} \qquad \begin{matrix} S_{c_2} = \underline{1\ 1\ 0\ 1\ 0\ 1\ 1\ 0} \Rightarrow \\ \swarrow \quad \searrow \\ c_{n-1}=0; \quad c_{n-2}=0; \end{matrix} \end{aligned}$$

$$n = 8 \text{ biți}; \quad A = +78; \quad B = -53;$$

$$\begin{aligned}
 |A|_2 &= 1001110 \Rightarrow |A|_{C_2} = 1001110 \Rightarrow A_{C_2} = 01001110 + \\
 |B|_2 &= \underbrace{0110101}_{7 \text{ biți}} \Rightarrow |B|_{C_2} = \underbrace{1001011}_{7 \text{ biți}} \Rightarrow B_{C_2} = 11001011 \\
 S_{C_2} &= \cancel{1}00011001 \Rightarrow \\
 &\Rightarrow S = +25; \quad c_{n-1}=1; \quad c_{n-2}=1;
 \end{aligned}$$

Observatii:

– În toate cazurile, se observă că depășirea domeniului poate fi determinată prin comparația transportului din rangul semnului c_{n-1} cu cel spre rangul semnului c_{n-2} :

Caz	Mărimi module	c_{n-1}	s_{n-1}	c_{n-2}	OVF
A, B pozitive	$ A + B < 2^{n-1}$	0	0	0	0
	$ A + B \geq 2^{n-1}$	0	1	1	1
A, B negative	$ A + B > 2^{n-1}$	1	0	0	1
	$ A + B \leq 2^{n-1}$	1	1	1	0
A pozitiv, B negativ	$ A < B $	0	1	0	0
	$ A \geq B $	1	0	1	0

$$OVF = \begin{cases} 0; & \text{dacă: } c_{n-1} = c_{n-2} \\ 1; & \text{dacă: } c_{n-1} \neq c_{n-2} \end{cases} \Rightarrow OVF = c_{n-1} \oplus c_{n-2}$$

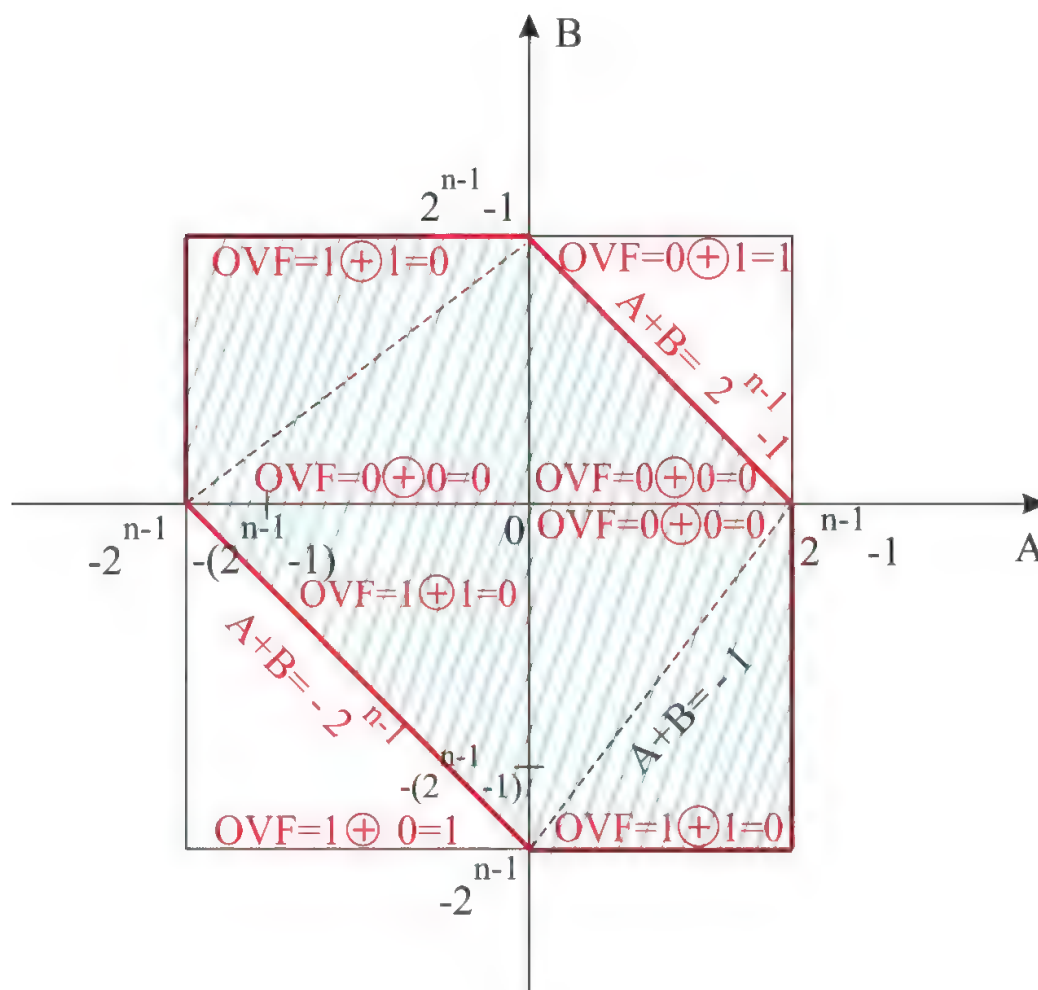
(comparație logică pe bit)

– Depășirea apare la numerele pozitive când suma modulelor atinge valoarea 2^{n-1} , pe când la cele negative numai peste această valoare (deoarece pentru numere negative este disponibil un cod în plus, conform C_2);

– După determinarea OVF , transportul c_{n-1} se ignoră;

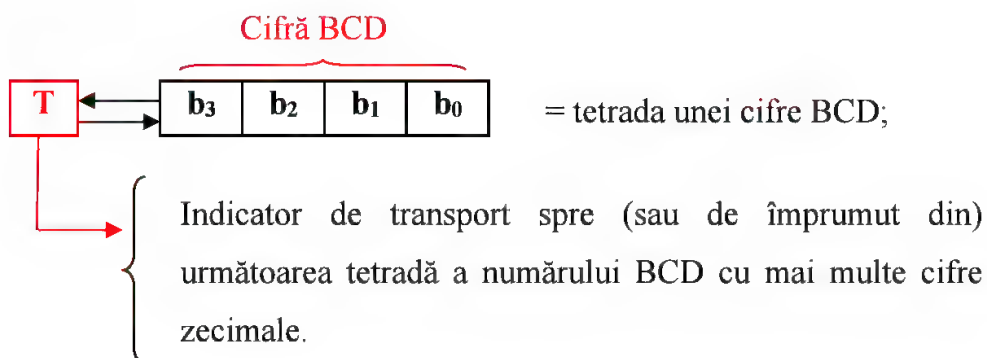
– Se semnalează depășirea prin OVF , dar nu se implementează hardware nici o corecție; aceasta rămâne în sarcina software-ului.

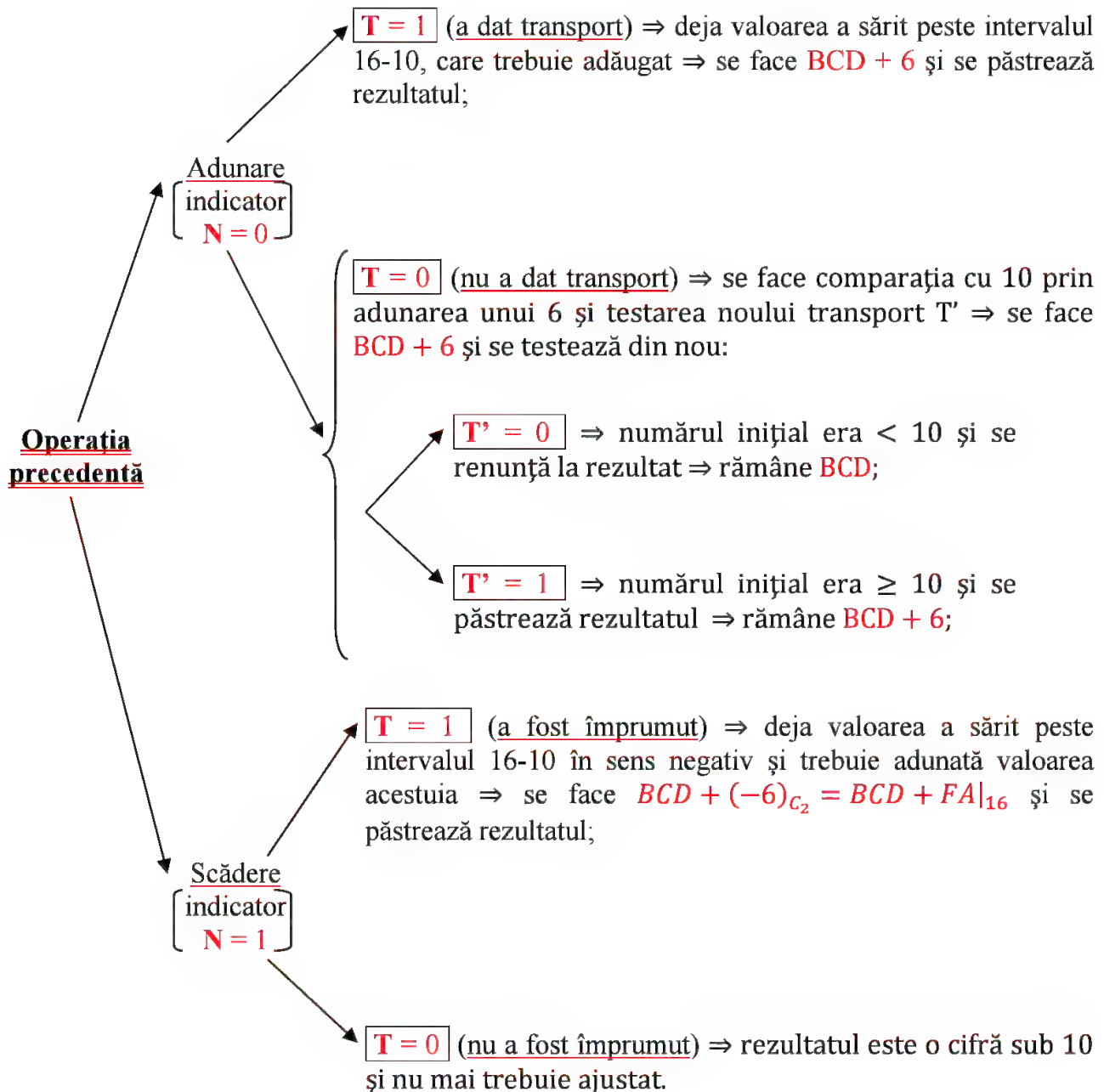
Domeniul permis pentru efectuarea calculelor cu reprezentări în C_2 fără a rezulta erori datorate depășirii se află în interiorul și pe marginile zonei hașurate:



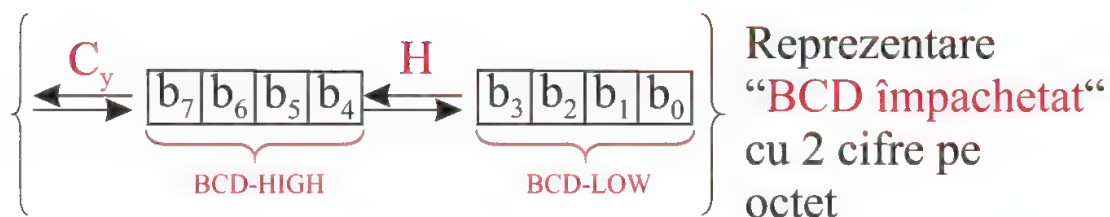
4.3. Operații cu numere reprezentate BCD

– După o operație cu tetrade **BCD** (adunare sau scădere) poate apărea necesitatea unei eventuale ajustări a rezultatului prin adăugarea sau scăderea suplimentară a unei constante egală cu diferența dintre cele două baze $(16 - 10) = 6$, pentru că au fost folosite doar primele 10 coduri ale bazei 16.





Observatii: – Dacă se lucrează cu reprezentări pe 8 biți (câte două cifre BCD) este necesar un indicator de transport între tetrade (**H** = “**HALF CARRY**”), iar pentru tetrada superioară se utilizează chiar indicatorul (**C_y** = “**CARRY**”):



– Ajustarea se face în cel mult 2 etape:

- { (I.) se adună $(\pm 06)_{16}$, se testează **H** și se păstrează sau nu rezultatul;
 (II.) se adună $(\pm 60)_{16}$, se testează **C_y** și se păstrează sau nu

Operație		Valori inițiale indicatori:		Valori indicatori după adunarea (± 66)			Valoare rezultat care se păstrează (ZZh = val. inițială)
N		C_y	H	Constanta	C_y	H	
0	ADUNARE	0	0	+66	0	0	ZZh + 00h
0		0	0	+66	0	1	ZZh + 06h
0		0	0	+66	1	0	ZZh + 60h
0		0	0	+66	1	1	ZZh + 66h
0		0	1	+66	0	–	ZZh + 06h
0		0	1	+66	1	–	ZZh + 66h
0		1	0	+66	–	0	ZZh + 60h
0		1	0	+66	–	1	ZZh + 66h
0		1	1	+66	–	–	ZZh + 66h
1		0	0	–00	–	–	ZZh + 00h
1	SCĂDERE	0	1	–06	–	–	ZZh + FAh
1		1	0	–60	–	–	ZZh + A0h
1		1	1	–66	–	–	ZZh + 9Ah
1		1	1	–66	–	–	ZZh + 9Ah

{ “–” = indiferent, nu contează dacă e 0 sau 1;
 (ZZ)₁₆ = ZZh = două cifre BCD în baza hexazecimal.

5. Algebra Booleană și algebra logică

– Algebra Booleană (introdusă de matematicianul *George Boole*) reprezintă cel mai important mijloc de analiză și sinteză a structurilor hardware.

Definiție: – O mulțime M ce conține cel puțin două elemente distincte formează o algebră Booleană împreună cu legile de compoziție (operațiile) notate “ $\cdot$ ” și “ $+$ ”, dacă:

A₁: – Mulțimea M este închisă față de cele două operații:

$$\forall a, b \in M \Rightarrow \begin{cases} a + b \in M; \\ a \cdot b \in M. \end{cases}$$

A₂: – În mulțimea M există câte un element neutru pentru fiecare dintre cele două operații:

$$\forall a \in M \Rightarrow \begin{cases} \exists "0" \in M \text{ a.î. } a + 0 = 0 + a = a; \\ \exists "1" \in M \text{ a.î. } a \cdot 1 = 1 \cdot a = a. \end{cases}$$

A₃: – În mulțimea M există complementul oricărui element prin operațiile considerate:

$$\forall a \in M \Rightarrow \exists "a" \in M \text{ a.î. } \begin{cases} a + \bar{a} = \bar{a} + a = 1; \\ a \cdot \bar{a} = \bar{a} \cdot a = 0. \end{cases}$$

(operația cu complementul conduce la elementul neutru pentru *cealaltă* operație).

A₄: – Cele două legi de compoziție sunt comutative:

$$\forall a, b \in M \Rightarrow \begin{cases} a + b = b + a; \\ a \cdot b = b \cdot a. \end{cases}$$

A₅: – Cele două operații sunt și asociative:

$$\forall a, b, c \in M \Rightarrow \begin{cases} a + (b + c) = (a + b) + c; \\ a \cdot (b \cdot c) = (a \cdot b) \cdot c. \end{cases}$$

A₆: – Operațiile sunt distributive una față de cealaltă:

$$\forall a, b, c \in M \Rightarrow \begin{cases} a \cdot (b + c) = (a \cdot b) + (a \cdot c); \\ a + (b \cdot c) = (a + b) \cdot (a + c). \end{cases}$$

(distributivitatea lui “ $+$ ” față de “ $\cdot$ ” este specifică doar algebrelor de tip Boolean).

Observații: – Mulțimea cea mai restrânsă care mai poate forma o algebră Booleană conține doar elementele neutre ale celor două operații: $M = \{0, 1\}$;

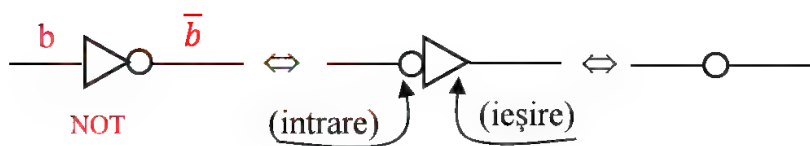
$$- \text{Dacă se asociază: } \left\{ \begin{array}{l} "0" = \text{propoziție falsă;} \\ "1" = \text{propoziție adevărată;} \\ "+" = \text{disjuncția logică SAU;} \\ "\cdot" = \text{conjuncția logică ȘI;} \end{array} \right\} \Rightarrow \text{se obține}$$

algebra logică, care este o algebră Booleană specială, cu 2 elemente și operatori logici.

5.1. Porti logice elementare

– Sunt structuri electronice care implementează fizic o operație elementară din algebra logică.

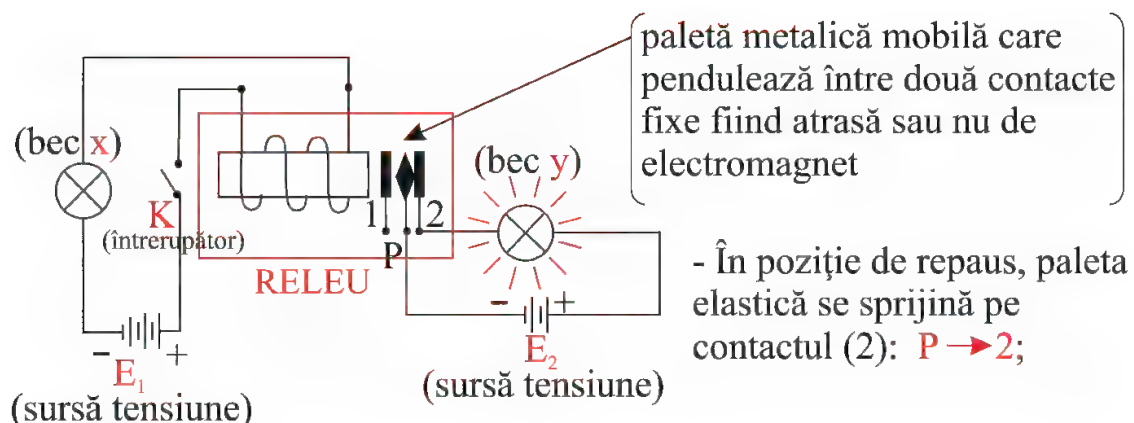
INVERSORUL (implementează complementarea):



b	$\bar{b}$
0	1
1	0

tabel de adevăr (descrie efectul operației).

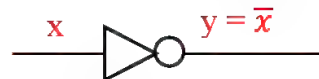
– Funcționarea inversorului poate fi simulată cu ajutorul unui circuit electric care conține un **RELEU ELECTROMAGNETIC**:



Dacă:

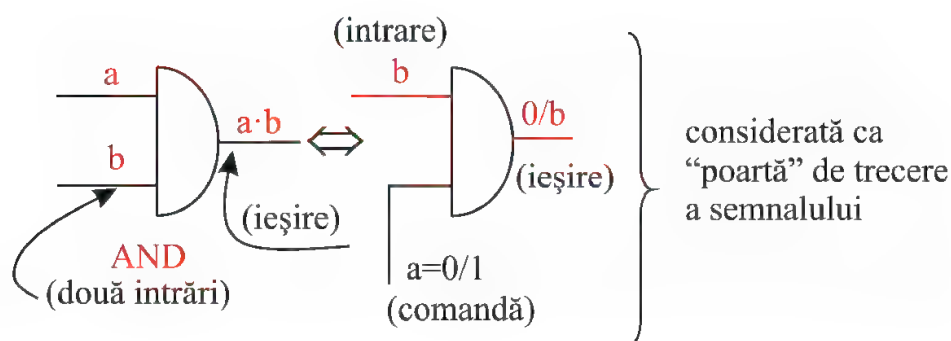
- întrerupătorul K este deschis $\Rightarrow$ prin circuitul $(E_1 \rightarrow K \rightarrow \text{bobină} \rightarrow \text{bec } x \rightarrow E_1)$ nu circulă curent și becul este stins ($x = 0$); paleta elastică se sprijină pe contactul (2) iar prin circuitul $(E_2 \rightarrow \text{bec } y \rightarrow 2 \rightarrow P \rightarrow E_2)$ circulă un curent care aprinde becul ($y = 1$).
- întrerupătorul K este închis $\Rightarrow$ prin circuitul $(E_1 \rightarrow K \rightarrow \text{bobină} \rightarrow \text{bec } x \rightarrow E_1)$ circulă curent care aprinde becul ($x = 1$) și alimentează bobina $\Rightarrow$ miezul se magnetizează și atrage paleta elastică în poziția contactului (1): $P \rightarrow 1$; prin circuitul $(E_2 \rightarrow \text{bec } y \rightarrow 2 \rightarrow P \rightarrow E_2)$ nu mai circulă curent iar becul se stinge ($y = 0$).

– Circuitul funcționează ca un inversor:



Observație: – Nu există nici un contact electric între circuitul becului de intrare și circuitul becului de ieșire; informația (semnalul) de la intrare se transmite la ieșire prin cuplaj magnetic.

– **Poarta logică ȘI** (implementează operația “ $\cdot$ ”):



$\backslash b$	0	1
a	0	0
1	0	1

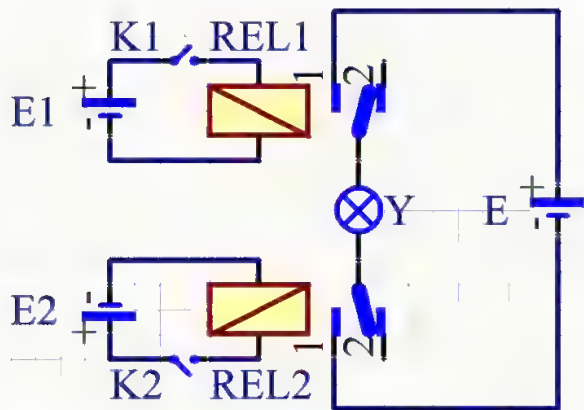
tabel de adevăr $\Leftrightarrow$

a	AND b
0	0
1	b

tabel compact de adevăr

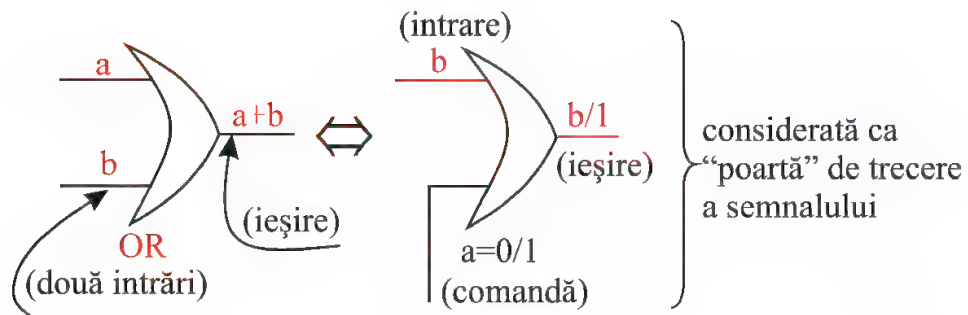
– Poarta ȘI permite trecerea semnalului de la intrare la ieșire numai când intrarea de comandă este pe “1”, altfel blochează ieșirea la “0”.

– Funcționarea porții ȘI poate fi simulată cu ajutorul unui circuit electric cu 2 rele care au contactele **P-(1)** conectate în serie față de circuitul becului de ieșire. Circuitele de intrare sunt complet izolate din punct de vedere electric, atât între ele cât și față de circuitul de ieșire; informația (semnalul) se transmite de la intrări la ieșire prin intermediul câmpului magnetic din bobinele releelor.



$\left\{ \begin{matrix} K_1 = a \\ K_2 = b \end{matrix} \right\} \Rightarrow$ becul se aprinde numai când K_1 **ȘI** K_2 sunt ambele închise (conduc), în rest este stins $\Rightarrow y = a \cdot b$;

– **Poarta logică SAU** (implementează operația “+”):



$\backslash b$	0	1
0	0	1
1	1	1

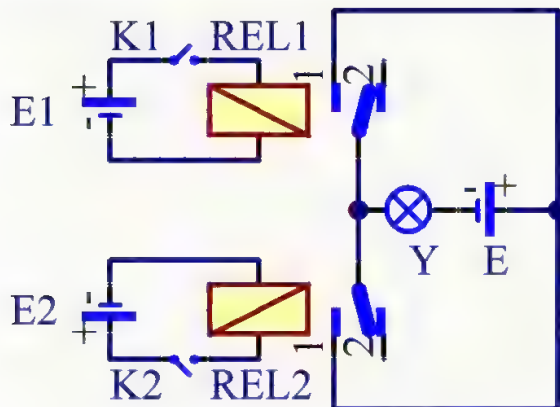
tabel de adevăr $\Leftrightarrow$

a	OR b
0	b
1	1

tabel compact de adevăr

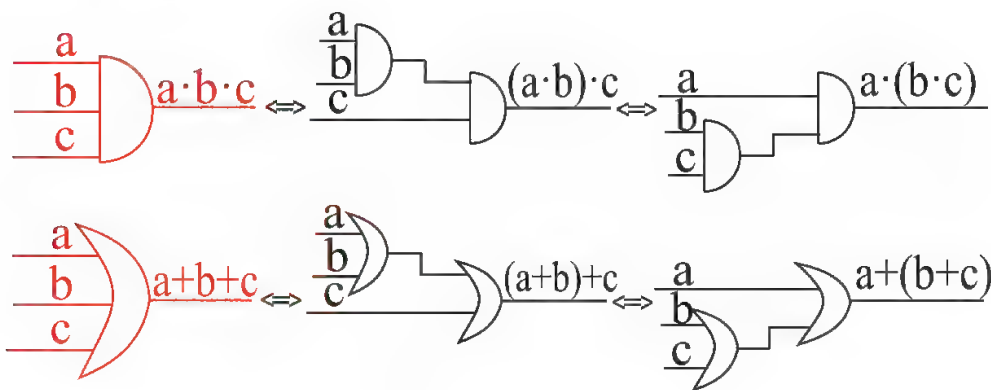
– Poarta SAU permite trecerea semnalului de la intrare la ieșire numai când intrarea de comandă este pe “0”, altfel blochează ieșirea la “1”.

– Funcționarea porții SAU poate fi simulată cu ajutorul unui circuit electric cu 2 rele care au contactele **P-(1)** conectate în paralel față de circuitul becului de ieșire. Toate circuitele de intrare și de ieșire sunt complet izolate din punct de vedere electric între ele; cuplajul semnalului de la fiecare intrare către ieșire se face prin câmpul magnetic al releelor.



$\begin{cases} K_1 = a \\ K_2 = b \end{cases} \Rightarrow$ becul se aprinde când
oricare dintre K_1 **SAU** K_2 este închis
(conduce), în rest este stins $\Rightarrow y = a + b$;

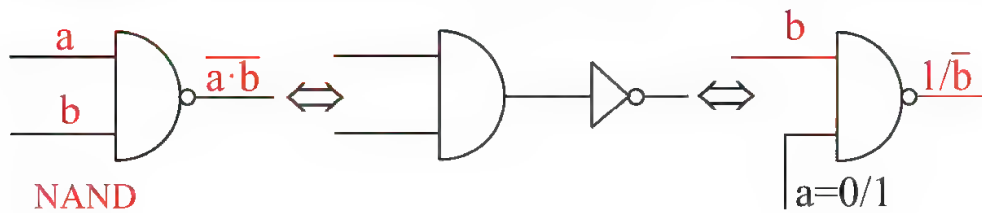
Observatie: – Conform asociativității, operațiile logice între mai multe elemente se pot implementa și consecutiv:



5.2. Porți logice ne-elementare

– Sunt structuri electronice care se pot implementa folosind porțile logice elementare, dar uneori pot fi implementate direct într-o variantă mai simplă decât cele elementare.

- **Poarta NUMAI (ȘI-NU)**: – La ieșire avem “0” numai când ambele intrări sunt “1”.

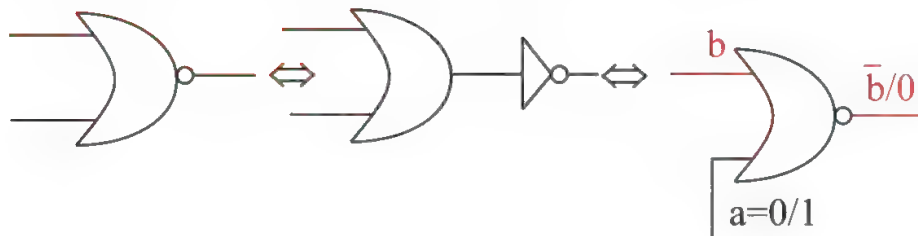


a \ b	0	1
0	1	1
1	1	0

COMPACTARE
(PE ORIZONTALĂ)

a	NAND b
0	1
1	$\bar{b}$

- **Poarta NICI (SAU-NU)**: – La ieșire avem “1” când niciuna din intrări nu este “1”.

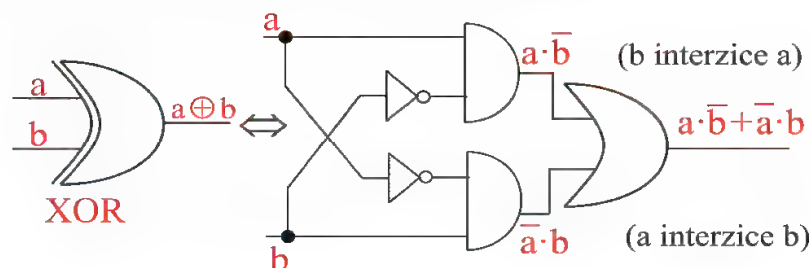


a \ b	0	1
0	1	0
1	0	0

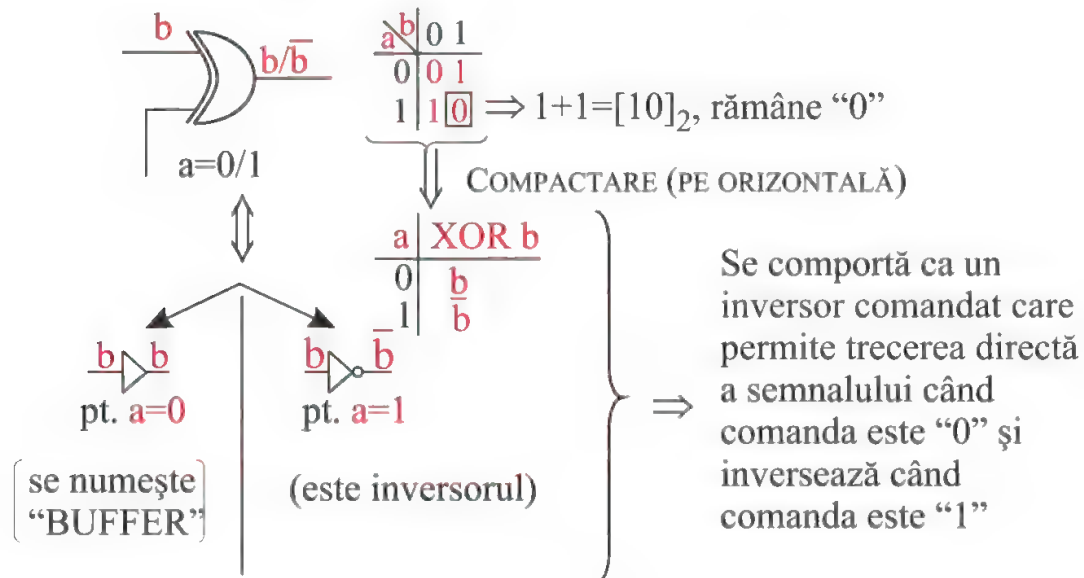
COMPACTARE
(PE ORIZONTALĂ)

a	NOR b
0	$\bar{b}$
1	0

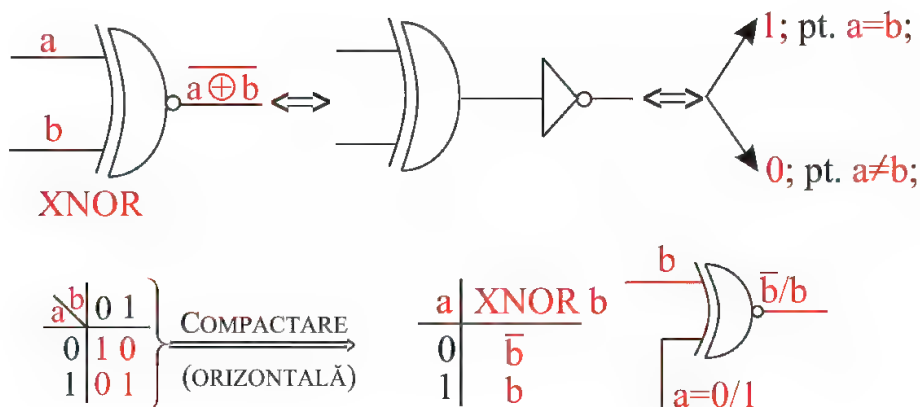
- **Poarta SAU-EXCLUSIV (SUMA MODULO 2)**: – La ieșire avem “1” când oricare dintre (a) sau (b) este “1”, exclusiv atunci când sunt ambele “1”;



- Este o funcție SAU între două operații de interdicție: (**b** interzice trecerea lui **a** $\Leftrightarrow a \cdot \bar{b}$) SAU (**a** interzice trecerea lui **b** $\Leftrightarrow \bar{a} \cdot b$), echivalentă cu suma pe un singur bit dintre **a** și **b**.



– **Poarta COINCIDENȚĂ (SAU-EXCLUSIV-NEGAT)**: – Se comportă ca un comparator pe un singur bit care are ieșirea "1" când intrările sunt identice și "0" când sunt diferite;

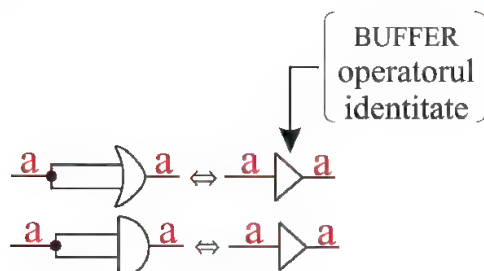


5.3. Teoremele algebrei logice

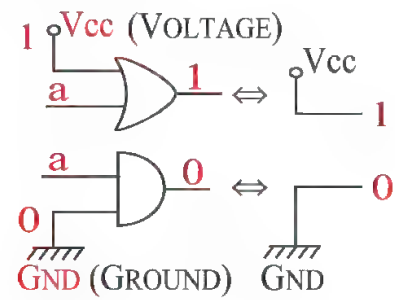
– Se pot demonstra cel mai comod folosind tabele de adevăr.

T₁: – IDEMPOTENȚA (tautologia):

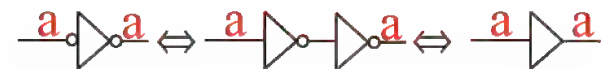
$$\forall a \in M \Rightarrow \begin{cases} a + a = a; \\ a \cdot a = a; \end{cases}$$



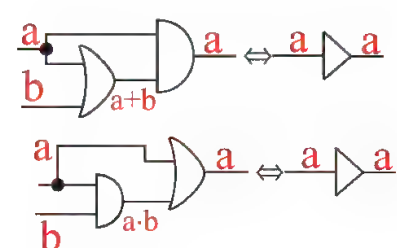
T₂: – Legile elementelor neutre (blocarea porților):

$$\forall a \in M \Rightarrow \begin{cases} a + 1 = 1 + a = 1; \\ a \cdot 0 = 0 \cdot a = 0; \end{cases}$$


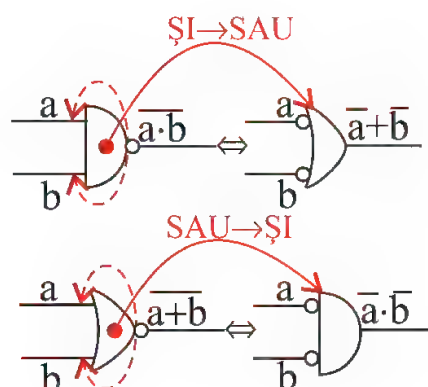
T₃: – INVOLUȚIA (dubla negație):

$$\forall a \in M \Rightarrow \overline{\overline{a}} = a;$$


T₄: – ABSORBȚIA:

$$\forall a, b \in M \Rightarrow \begin{cases} a \cdot (a + b) = a; \\ a + (a \cdot b) = a; \end{cases}$$


T₅: – Teoremele lui DE MORGAN:

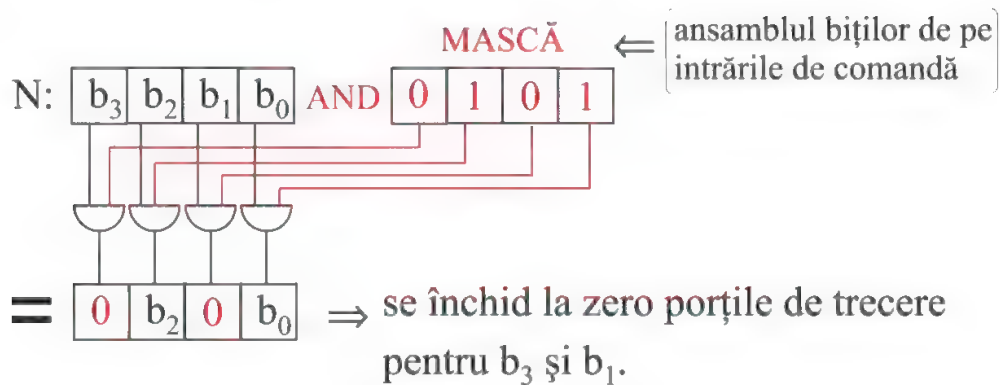
$$\forall a, b \in M \Rightarrow \begin{cases} \overline{a \cdot b} = \overline{a} + \overline{b}; \\ \overline{a + b} = \overline{a} \cdot \overline{b}; \end{cases}$$


– Se poate “transla” cerculețul de negare prin poartă, de la ieșire spre ambele intrări, cu transformarea porții în “duala” ei:

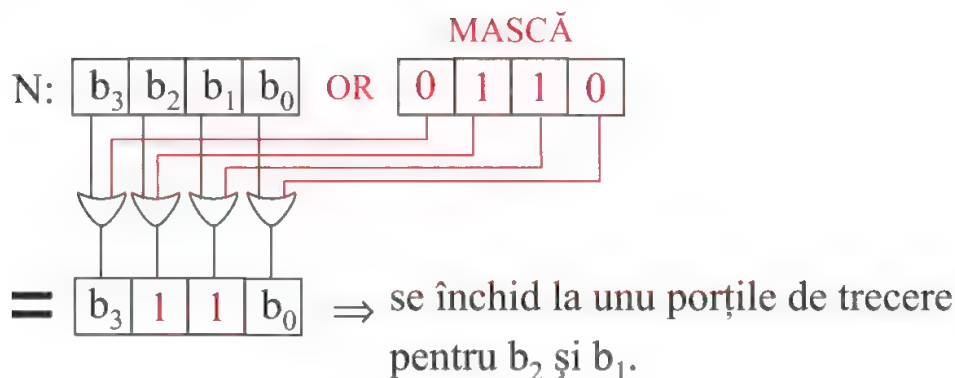
$$\text{ȘI} \xleftrightarrow{\text{O}} \text{SAU}$$

5.4. Operații logice pe mai mulți biți

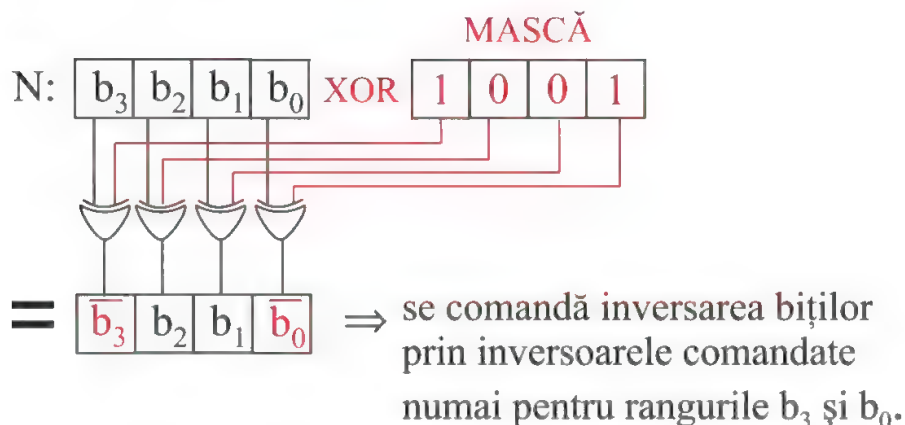
– Pentru forțarea selectivă la “0” a unor biți dintr-un număr se folosește operația **AND** cu o mască care are “0” în pozițiile de forțat și “1” în rest:



– Pentru forțarea selectivă la “1” a unor biți dintr-un număr se folosește operația **OR** cu o mască care are “1” în pozițiile de forțat și “0” în rest:

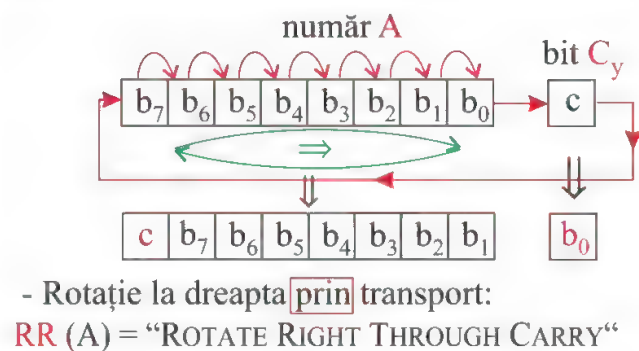
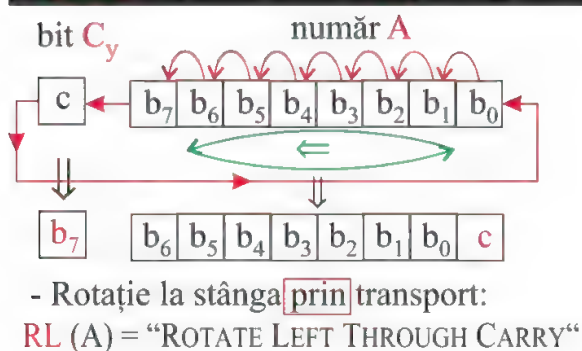
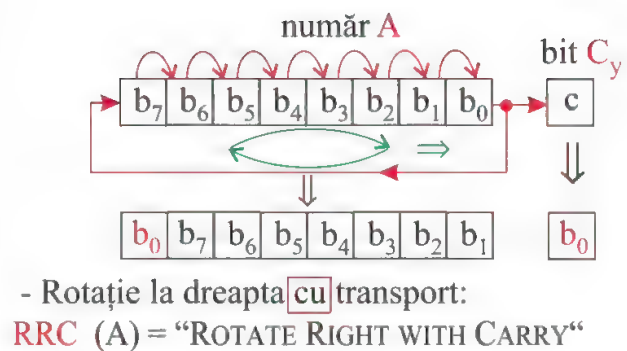
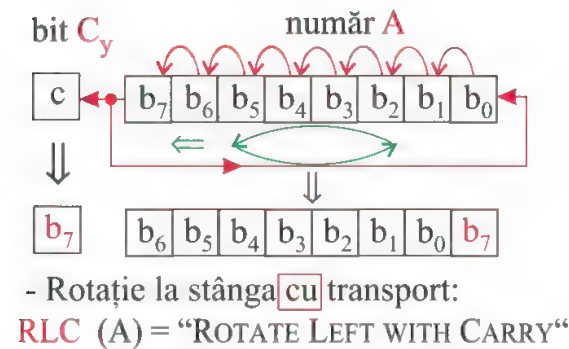


– Pentru inversarea selectivă a unor biți dintr-un număr se folosește operația **XOR** cu o mască care are “1” în pozițiile de inversat și “0” în celelalte:



5.5. Operații aritmetico – logice de rotație

– Produc deplasarea simultană a ansamblului de biți spre stânga sau spre dreapta, cu recircularea bitului de la margine în partea cealaltă:



– Pentru înțelegere, rotațiile pot fi asemănată cu deplasarea șenilei unui tanc cu țeava în direcția de rotație, dată de bitul **C_y**, prin care se “trage” cu biți.

Observații: – Operațiile de rotație pot fi transformate simplu în operații de deplasare prin aducerea la zero a bitului care a intrat din partea cealaltă.

$$\begin{array}{l}
 \textcolor{red}{SL}(A) = \left\{ \begin{array}{l} \textcolor{red}{RLC}(A) \\ \textcolor{red}{AND} \text{ 11111110} \end{array} \right. \quad \left| \quad \textcolor{red}{SR}(A) = \left\{ \begin{array}{l} \textcolor{red}{RRC}(A) \\ \textcolor{red}{AND} \text{ 01111111} \end{array} \right. \right. \\
 \text{(SHIFT LEFT)} \qquad \qquad \qquad \text{(SHIFT RIGHT)}
 \end{array}$$

– Operațiile de deplasare a cifrelor unui număr $\boxed{N}$ scris într-o bază $\boxed{B}$ spre stânga (respectiv spre dreapta) cu o poziție sunt echivalente cu înmulțirea (respectiv împărțirea) numărului prin $10|_B$ scris în baza B , adică cu valoarea bazei B în zecimal.

Demonstrație: $N = \overleftarrow{(a_{n-1}a_{n-2} \dots a_1a_0, a_{-1} \dots a_{-(m-1)}a_{-m})_B};$

$$N|_{10} = a_{n-1} \cdot B^{n-1} + a_{n-2} \cdot B^{n-2} + \dots + a_1 \cdot B^1 + a_0 \cdot B^0 + a_{-1} \cdot B^{-1} + \dots + a_{-(m-1)} \cdot B^{-(m-1)} + a_{-m} \cdot B^{-m};$$

$$\begin{aligned} N \cdot B|_{10} &= \cancel{a_{n-1} B^n} + a_{n-2} \cdot B^{n-1} + \dots + a_1 \cdot B^2 + a_0 \cdot B^1 + a_{-1} \cdot B^0 + \\ &+ a_{-2} \cdot B^{-1} + \dots + a_{-(m-1)} \cdot B^{-(m-2)} + a_{-m} \cdot B^{-(m-1)} + \boxed{0} \cdot B^{-m} \Rightarrow \\ &\left(\begin{array}{l} \text{cifra } \boxed{a_{n-1}} \text{ se pierde deoarece} \\ \text{nu încapă pe } \boxed{n} \text{ biți} \end{array} \right) \Rightarrow N \cdot B = (a_{n-2}a_{n-3} \dots a_0a_{-1}, a_{-2} \dots a_{-m} \boxed{0})_B \\ &= SL(N); \end{aligned}$$

$$\begin{aligned} N \cdot B^{-1}|_{10} &= \boxed{0} \cdot B^{n-1} + a_{n-1} \cdot B^{n-2} + a_{n-2} \cdot B^{n-3} + \dots + a_2 \cdot B^1 + a_1 \cdot B^0 + \\ &+ a_0 \cdot B^{-1} + \dots + a_{-(m-1)} \cdot B^{-m} + \cancel{a_{-m} \cdot B^{-(m+1)}} \Rightarrow \\ &\left(\begin{array}{l} \text{cifra } \boxed{a_{-m}} \text{ se pierde deoarece} \\ \text{se lucrează cu } \boxed{m} \text{ biți} \\ \text{după virgulă} \end{array} \right) \Rightarrow N \cdot B^{-1} = (\boxed{0} a_{n-2} \dots a_2 a_1, a_0 \dots a_{-(m-1)})_B = \\ &= SR(N); \end{aligned}$$

6. Formele canonice ale funcțiilor logice

6.1. Forma canonică disjunctivă (FCD)

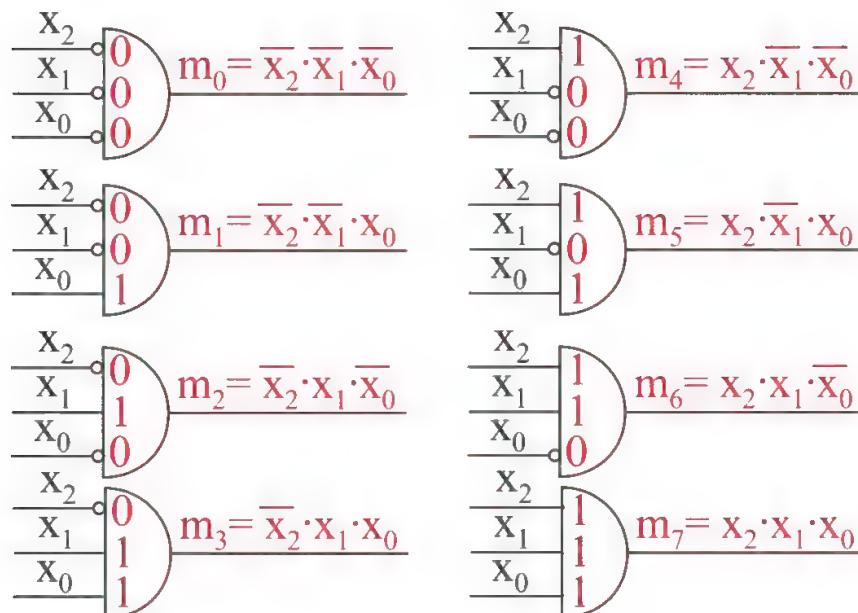
– Se numește constituent al **unității** (**min-term**, termen minimal) pentru o funcție logică fiecare termen-produs dintre toate variabilele funcției, care poate lua valoarea “1” numai pentru o singură combinație a intrărilor:

m_k poate lua valoarea “1” numai când combinația binară de intrare are drept echivalent zecimal indicele **k** .

Exemplu: $y = f(x_2, x_1, x_0)$ = funcție de 3 variabile;

$$m_5 = 1 \Leftrightarrow (x_2 x_1 x_0)_2 = 5 \Leftrightarrow \begin{cases} x_2 = 1; \\ x_1 = 0; \\ x_0 = 1; \end{cases} \Rightarrow m_5 = x_2 \cdot \bar{x}_1 \cdot x_0;$$

în scrierea lui **m_k** variabilele care iau valoarea “1” se scriu simplu, iar cele care iau valoarea “0” se scriu negate.



– Forma canonică disjunctivă se scrie ca o sumă între toți min-termii corespunzători valorilor de **unu** din tabelul de adevăr al funcției:

$$\text{FCD: } y = \sum_{k=0}^{2^n-1} m_k \cdot \boxed{y_k} = \sum_{y_k=1} m_k$$

valorile funcției y pentru toate
combinațiile k din tabelul de adevăr

Exemplu: – Funcția de 3 variabile definită prin:

Tabel de adevăr			
x_2	x_1	x_0	$y = f(x_2, x_1, x_0)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$\Leftrightarrow$ Combinațiile zecimale ale intrărilor
pentru care funcția ia valoarea “1”

$$\boxed{y = R_1(1, 4, 5)}$$

$$\text{FCD: } y = m_1 + m_4 + m_5 =$$

$$= \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0$$

$$\begin{array}{ccc} \downarrow & \downarrow & \downarrow \\ (0 & 0 & 1)_2=1; (1 & 0 & 0)_2=4; (1 & 0 & 1)_2=5; \end{array}$$

6.2. Forma canonică conjunctivă (FCC)

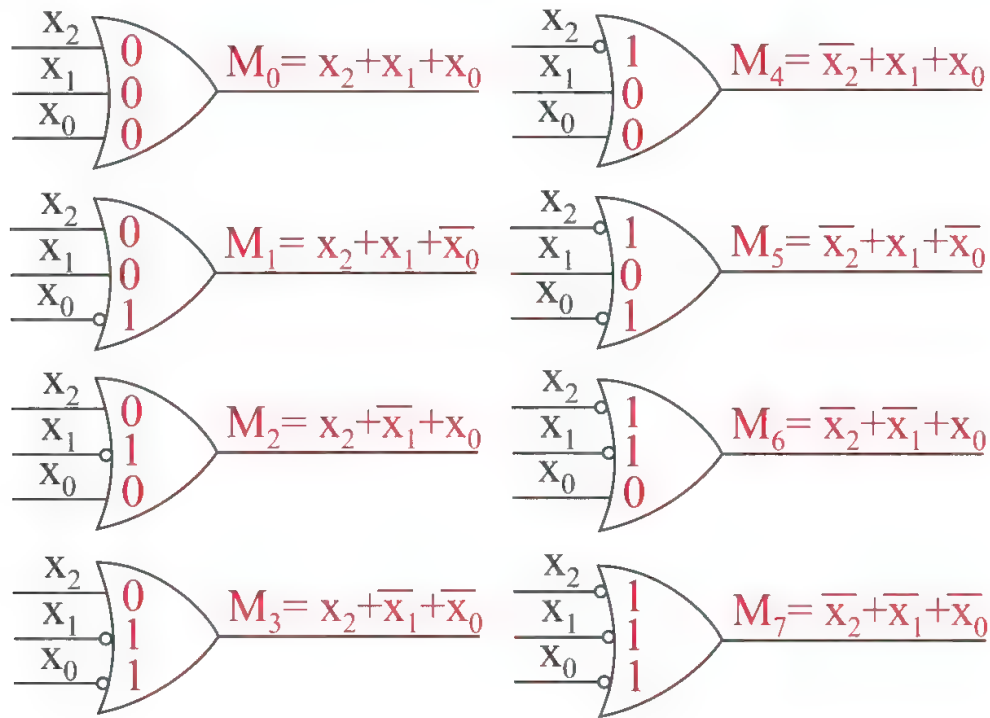
– Se numește constituent al lui **zero** (**max-term**, termen maximal) pentru o funcție logică fiecare termen sumă dintre toate variabilele funcției, care poate lua valoarea “0” numai pentru o singură combinație a intrărilor:

M_k poate lua valoarea “0” numai când combinația binară de intrare are drept echivalent zecimal indicele k .

Exemplu: $y = f(x_2, x_1, x_0)$ = funcție de 3 variabile;

$$M_2 = 0 \Leftrightarrow (x_2 \ x_1 \ x_0)_2 = 2 \Leftrightarrow \left\{ \begin{array}{l} x_2 = 0; \\ x_1 = 1; \\ x_0 = 0; \end{array} \right. \Rightarrow M_2 = x_2 + \overline{x_1} + x_0;$$

[în scrierea lui M_k variabilele care iau valoarea “0” se scriu
simplu, iar cele care iau valoarea “1” se scriu negat.]



Observație: – Conform teoremelor lui De Morgan, max-termul corespunzător unei combinații este egal cu min-termul **negat** al acelei combinații.

$$M_k = \overline{m_k}; \quad k = \overline{0, 2^n - 1} \text{ pentru } [n] \text{ variabile}$$

– Forma canonică conjunctivă se scrie ca un produs între toți max-termii corespunzători valorilor de **zero** din tabelul de adevăr al funcției:

$$\text{FCC: } y = \prod_{k=0}^{2^n-1} (M_k + y_k) = \prod_{y_k=0} M_k$$

Exemplu: – Funcția anterioară $y = f(x_2, x_1, x_0) = R_1(1, 4, 5) = \boxed{R_0(0, 2, 3, 6, 7)}$; (se definește prin combinațiile zecimale în care funcția ia valoarea “0”).

$$\begin{aligned} \text{FCC: } y = M_0 \cdot M_2 \cdot M_3 \cdot M_6 \cdot M_7 &= (x_2 + x_1 + x_0) \cdot (x_2 + \overline{x_1} + x_0) \cdot (x_2 + \overline{x_1} + \overline{x_0}) \cdot \\ &\quad \cdot (\overline{x_2} + \overline{x_1} + x_0) \cdot (\overline{x_2} + \overline{x_1} + \overline{x_0}) \end{aligned}$$

Observații: – Orice funcție logică poate fi exprimată complet folosind doar două niveluri de porți și un nivel de inversare;

– Dacă termenul ia o anumită valoare, variabilele care iau **aceeași valoare** vor apărea **simple** iar cele care iau valoarea opusă vor apărea negate.

7. Minimizarea grafică a funcțiilor logice

- Funcțiile logice pot fi aduse la forma cea mai simplă, ce utilizează un număr **minim de porți** care au un număr **minim de intrări**.
- Folosirea axiomelor și teoremelor algebrei logice pentru minimizare nu este o metodă sistematică, algoritmică și nici nu se știe întotdeauna dacă s-a ajuns la expresia optimă.

7.1. Diagrame Veitch – Karnaugh

Pentru minimizarea algoritmică a funcțiilor de până la 5 variabile se folosește metoda diagramelor grafice **VEITCH – KARNAUGH**.

– Diagrama VK = reprezentare matriceală a tabelului de adevăr în care combinațiile de intrare sunt **grupate în ordinea codului GRAY** $\Rightarrow$

$\Rightarrow$ două celule vecine diferă numai printr-o singură variabilă $\Rightarrow$

$\Rightarrow$ grupând mai multe celule vecine se vor elimina toate variabilele care își schimbă valoarea între celulele grupării.

VK pt. FCD de 4 var.

$X_1X_0 \backslash X_3X_2$	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	1 m_5	1 m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	1 m_9	1 m_{11}	m_{10}

(se trec valorile de "1" ale funcției corespunzătoare min-termilor)

VK pt. FCC de 4 var.

$X_1X_0 \backslash X_3X_2$	00	01	11	10
00	M_0	M_1	M_3	M_2
01	M_4	0 M_5	0 M_7	M_6
11	M_{12}	M_{13}	M_{15}	M_{14}
10	M_8	0 M_9	0 M_{11}	M_{10}

(se trec valorile de "0" ale funcției corespunzătoare max-termilor)

Exemplu: FCD:

$$f(x_3, x_2, x_1, x_0) = \underbrace{(m_5 + m_7)}_{\substack{\text{adiacenți} \\ G_1}} + \underbrace{(m_9 + m_{11})}_{\substack{\text{adiacenți} \\ G_2}}$$

$$\begin{aligned} m_5 + m_7 &= \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \cdot x_0 + \bar{x}_3 \cdot x_2 \cdot x_1 \cdot x_0 \\ &= \bar{x}_3 \cdot x_2 \cdot \underbrace{(x_1 + \bar{x}_1)}_{=1} \cdot x_0 = \bar{x}_3 \cdot x_2 \cdot x_0 \Rightarrow \end{aligned}$$

$\Rightarrow$ a dispărut variabila x_1 care se modifică în cadrul grupării G_1 .

FCC:

$$f(x_3, x_2, x_1, x_0) = \underbrace{(M_5 \cdot M_7)}_{\substack{\text{adiacenți} \\ G_1}} \cdot \underbrace{(M_9 \cdot M_{11})}_{\substack{\text{adiacenți} \\ G_2}}$$

$$\begin{aligned} M_9 \cdot M_{11} &= (\bar{x}_3 + x_2 + x_1 + \bar{x}_0) \cdot \\ &\cdot (\bar{x}_3 + x_2 + \bar{x}_1 + \bar{x}_0) = [(\bar{x}_3 + x_2 + \bar{x}_0) + x_1] \cdot \\ &\cdot [(\bar{x}_3 + x_2 + \bar{x}_0) + \bar{x}_1] = \\ &= (z + x_1) \cdot (z + \bar{x}_1) = z + z \cdot \bar{x}_1 + \\ &+ z \cdot x_1 + \underbrace{x_1 \cdot \bar{x}_1}_{=0} = z + z \cdot \underbrace{(x_1 + \bar{x}_1)}_{=1} = z \Rightarrow \\ &\Rightarrow M_9 \cdot M_{11} = \bar{x}_3 + x_2 + \bar{x}_0. \end{aligned}$$

Observații: – Termenii **adiacenți** sunt cei alăturați în tabel, dar și cei dispuși simetric față de liniile mediane ale diagramei (reprezentarea este desfășurarea plană a unui hipercub de dimensiune 4).

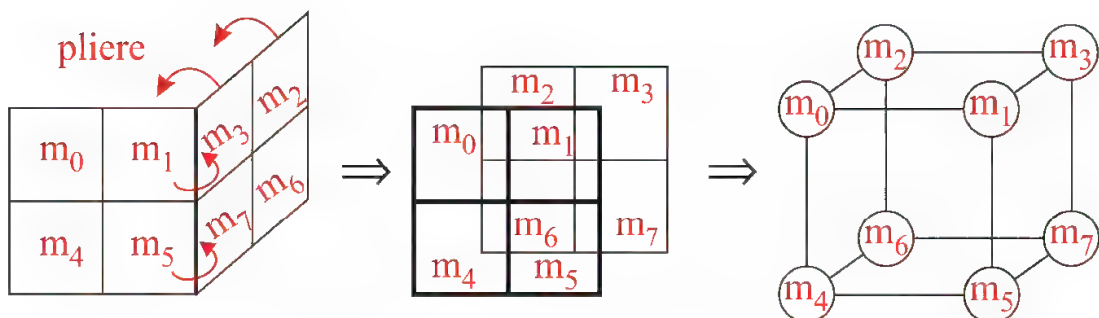
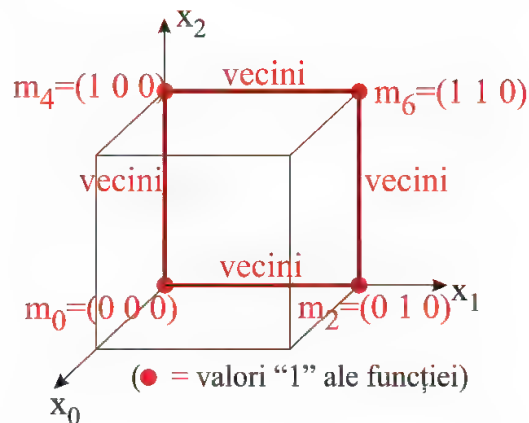
– Se pot grupa numai un număr de $[2^0, 2^1, 2^2, \dots, 2^p]$ celule adiacente (putere a lui 2) $\Rightarrow$ ceea ce conduce la dispariția a $[0, 1, 2, \dots, p]$ variabile din **FCD** sau **FCC**.

Exemplu:

$x_1 \backslash x_0$	0 0	0 1	1 1	1 0
$x_2 \backslash$				
0	1			1
1	1			1

G_1 ; mediană

$$f(x_2, x_1, x_0) = \bar{x}_0;$$



Exemplu: $y = R_0(3, 4, 5, 7, 11, 13);$

$x_1 \ x_0$	0 0	0 1	1 1	1 0
$x_3 \ x_2$				
0 0			0	
0 1	0	0	0	
1 1		0		
1 0			0	

Diagram illustrating the Karnaugh map for the function $y = R_0(3, 4, 5, 7, 11, 13)$. The map shows four groups of 1s (circled in green) labeled G_1 , G_2 , G_3 , and G_4 . G_1 is a vertical group of three cells (01, 11, 10) in column 01. G_2 is a horizontal group of three cells (00, 01, 11) in row 01. G_3 is a vertical group of two cells (00, 10) in column 11. G_4 is a horizontal group of two cells (00, 10) in row 00. A label "mediană verticală" points to the vertical line between columns 01 and 11, and "mediană orizontală" points to the horizontal line between rows 00 and 01.

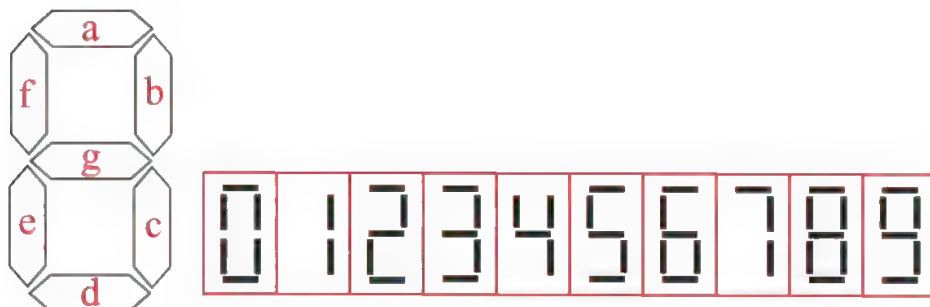
– Se face un număr cât mai mic de grupări, fiecare incluzând cât mai multe celule, astfel încât să fie prinse toate celulele.

$$y = \underbrace{(\overline{x_2} + x_1 + \overline{x_0})}_{G_1 \text{ (dispare } x_3)} \cdot \underbrace{(x_3 + \overline{x_1} + \overline{x_0})}_{G_2 \text{ (dispare } x_2)} \cdot \underbrace{(x_3 + \overline{x_2} + x_1)}_{G_3 \text{ (dispare } x_0)} \cdot \underbrace{(x_2 + \overline{x_1} + \overline{x_0})}_{G_4 \text{ (dispare } x_3)}.$$

7.2. Funcții logice incomplet definite

- Unele dintre cele 2^n combinații posibile ale variabilelor de intrare pentru o funcție de n variabile pot să nu apară niciodată în realitate;
- Pentru aceste combinații funcția rămâne nedefinită, fiind definită **incomplet** pe domeniul celor 2^n combinații;
- Ieșirea se va considera indiferentă în tabelul de adevăr pentru toate combinațiile imposibile (adică poate fi sau "0" sau "1").

Exemplu: – Funcțiile segmentelor unui afișaj cu 7 segmente pentru cifre zecimale codificate în BCD pe 4 biți:



$$g(x_3, x_2, x_1, x_0) = R_1(2, 3, 4, 5, 6, 8, 9) + R_{-}(10, 11, 12, 13, 14, 15)$$

valori indiferente pentru că nu apar
în codul unei cifre BCD

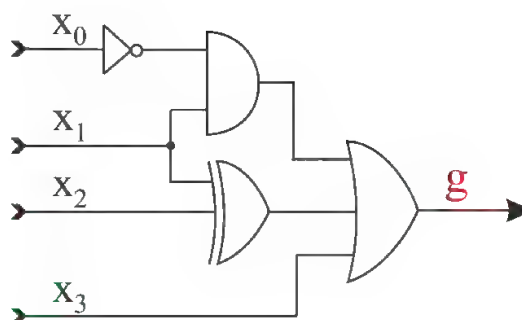
– La implementare, valorile indiferente se vor considera fie “0”, fie “1”, în funcție de avantajele pe care le pot aduce la constituirea unor grupări mai mari în minimizarea funcției.

$$g(x_3, x_2, x_1, x_0) = x_3 + \underbrace{(x_2 \cdot \bar{x}_1 + \bar{x}_2 \cdot x_1)}_{\text{(XOR)}} + x_1 \cdot \bar{x}_0 = x_3 + (x_1 \oplus x_2) + x_1 \cdot \bar{x}_0 ;$$

$x_1 \ x_0$	0 0	0 1	1 1	1 0
$x_3 \ x_2$				
0 0	0	0	1	1
0 1	1	1	0	1
1 1	-	-	-	-
1 0	1	1	-	-

(“ - ” = combinații indiferente)

G_1 G_2 G_3 G_4

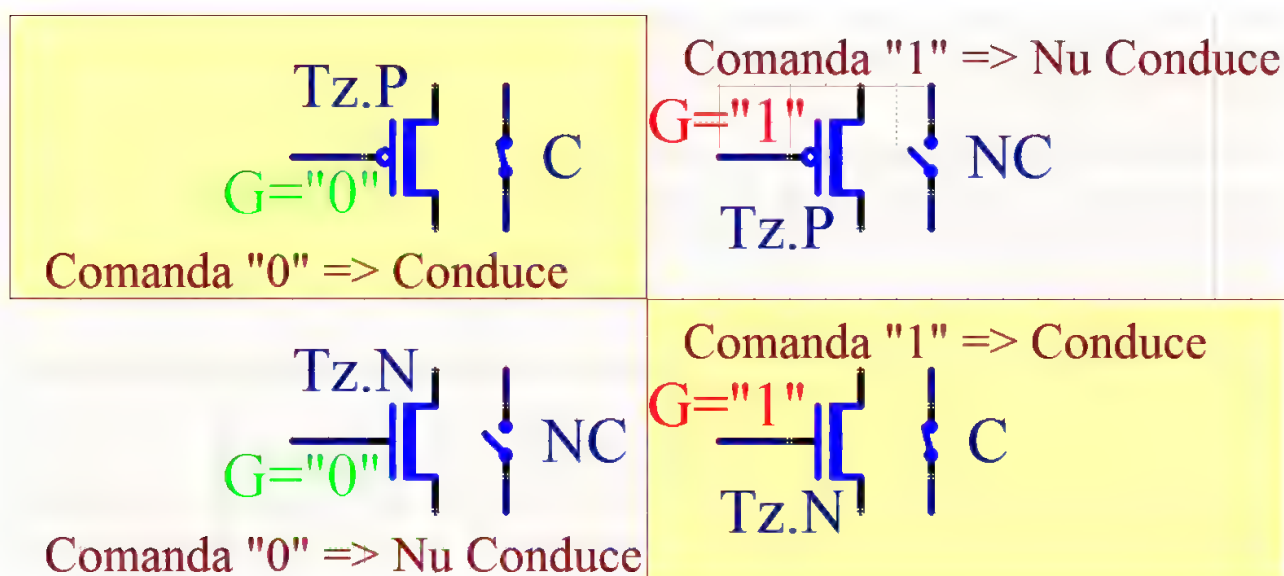


8. Etajul de ieșire al porților logice; magistrale

– În prezent, implementarea fizică a structurilor logice se realizează folosind **tranzistoare** care lucrează ca niște **comutatoare comandate** prin valori logice, asemănătoare ca principiu de funcționare releelor electromagnetice, cu intrarea de comandă (denumită **grilă** sau poartă) complet izolată din punct de vedere electric față de calea (circuitul) comutatorului comandat.

– Dacă la relee cuplajul între intrare și circuitul comandat se face prin câmp magnetic, la tranzistoarele utilizate în tehnica digitală cuplajul se face prin **câmp electric**. Câmpul produs de grila metalică asigură formarea unui canal conductor între capetele comutatorului semiconductor prin inducția unor purtători de sarcină, fără a avea un contact electric cu ele (grila este izolată de restul ansamblului printr-un oxid complet neconductor). De aceea, astfel de tranzistoare se mai numesc *“cu canal indus prin efect de câmp”*, având construcție **MOS** = *“Metal-Oxid-Semiconductor”*.

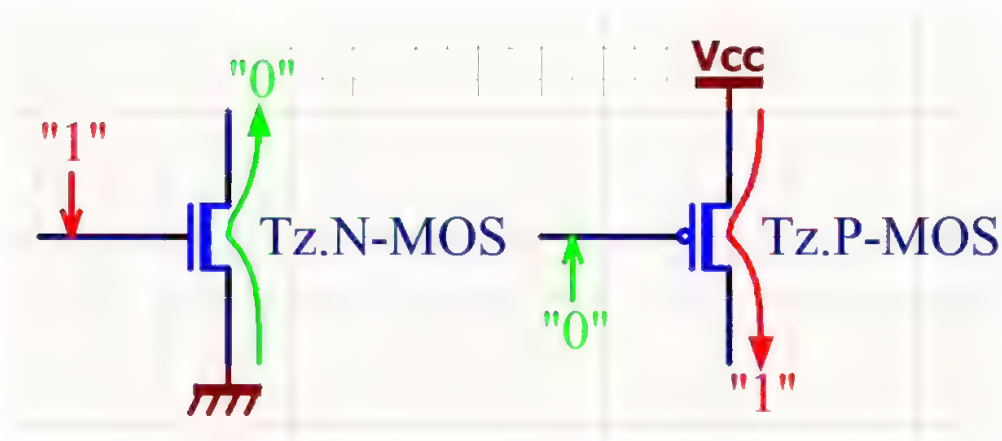
– După felul purtătorilor de sarcină care formează canalul de conducție, tranzistoarele **MOS** pot fi de două tipuri diferite: **Tz. de tip N** (cu canal **negativ**, conducție prin electroni) sau **Tz. de tip P** (cu canal **pozitiv**, conducție prin “goluri” = lipsa electronilor). Valoarea logică aplicată pe grila de comandă are efecte diferite asupra conducției, în funcție de tipul tranzistorului:



– Valoarea logică aplicată pe grila de comandă are doar rolul de a influența “de la distanță” starea de conducție a comutatorului (**închis** = Conduce, **deschis** = Nu Conduce), dar nu poate trece direct către nici unul din capetele sale. Prin comutator vor trece sau nu valori de “0” sau de “1” logic de la polii sursei de alimentare a sistemului (**Vcc** = polul **pozitiv**, sursa de “1”, **Gnd** = masa, polul **negativ**, sursa de “0”) și NU de la intrare !

– Avantajele tranzistoarelor față de relee sunt semnificative: mult mai mici, ele vor consuma foarte puțin curent; nu au părți mecanice în mișcare, deci nu au uzură și nici inerție; din aceste motive **pot comuta la viteze net superioare un timp nedefinit**, fără uzură fizică.

– Totuși, tranzistoarele au și un dezavantaj serios față de relee, care complică implementarea structurilor logice: spre deosebire de contactele de comutare ale releelor, prin care pot trece ambele valori logice “0” sau “1”, **prin canalul unui tranzistor nu poate trece decât una dintre cele două valori logice**, în funcție de tipul său. Tranzistorul de tip **N** nu poate conduce prin canalul său negativ decât “0” logic de la polul negativ al sursei (**Gnd**), iar prin tranzistorul de tip **P** cu canal pozitiv nu poate trece decât “1” logic de la polul pozitiv al sursei (**Vcc**):

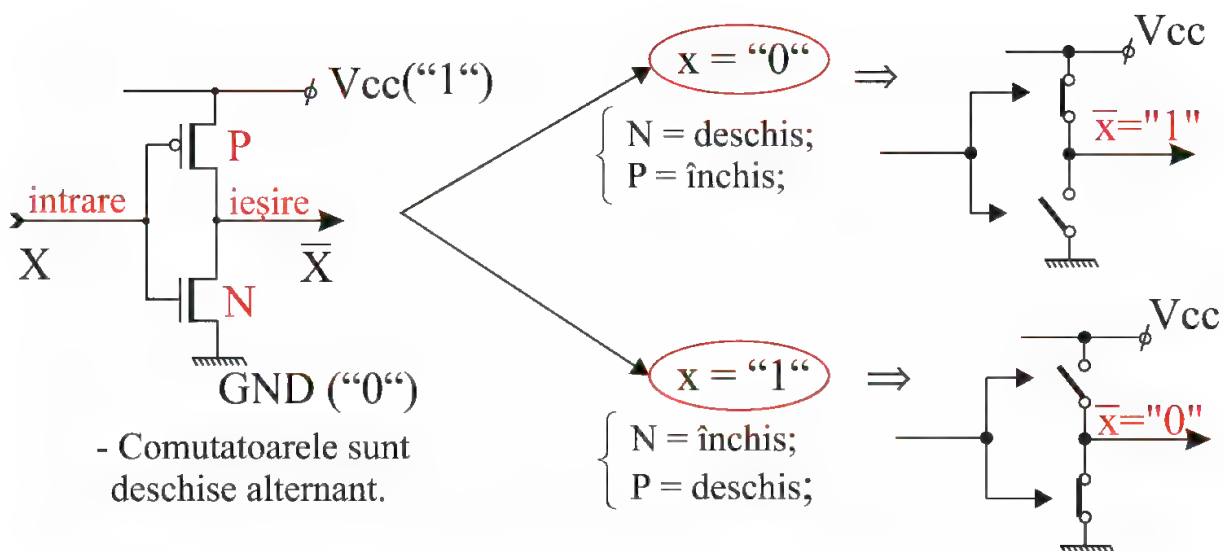


Observatii:

- Tz. **N** conduce “0” de la **Gnd** când primește comanda “1”;
- Tz. **N** nu conduce deloc când primește comanda “0”;
- Tz. **P** conduce “1” de la **Vcc** când primește comanda “0”;
- Tz. **P** nu conduce deloc când primește comanda “1”;

– În alcătuirea unei structuri logice trebuie să se regăsească ambele tipuri de tranzistoare: prin cele de tip N vor trece valorile de “0”, iar prin cele de tip P vor trece valorile de “1”. Etajul de ieșire al oricărei porți logice este format prin conectarea în serie a două tranzistoare P și N între polii sursei de alimentare, cu linia de ieșire preluată din punctul de legătură dintre ele.

Exemplu: – Inversor simplu:



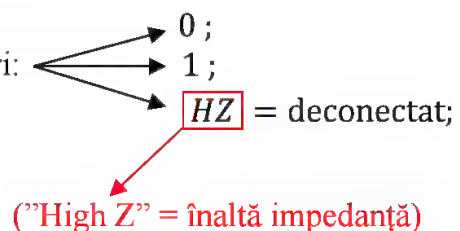
Observații: – Dintre toate porțile logice, inversorul are alcătuirea cea mai simplă (cu numai 2 tranzistoare) care comută și cel mai rapid; un “buffer”, chiar dacă nu modifică semnalul din punctul de vedere al valorii logice (are doar rolul de a izola intrarea de ieșire), se construiește folosind două inversoare consecutive, deci este mai complicat și mai lent;

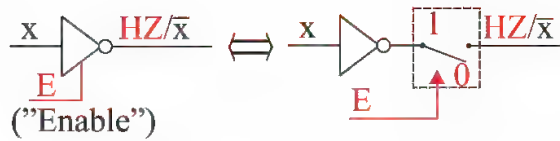
– Pot exista porți care să aibă la ieșire și starea în care ambele tranzistoare nu conduc, ieșirea dintre ele fiind practic deconectată;

– Nu pot conduce ambele tranzistoare simultan, din cauză că ar produce scurt-circuit între punctele de alimentare.

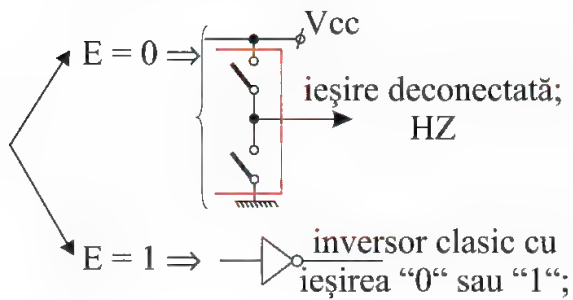
Exemplu:

– Inversoare și “buffer”-e cu 3 stări:



CU DECONECTARE PE “0” :

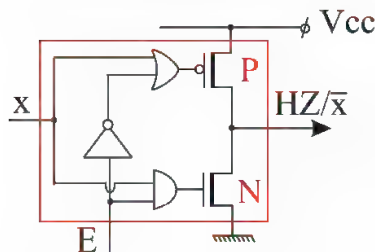
– Funcționare:



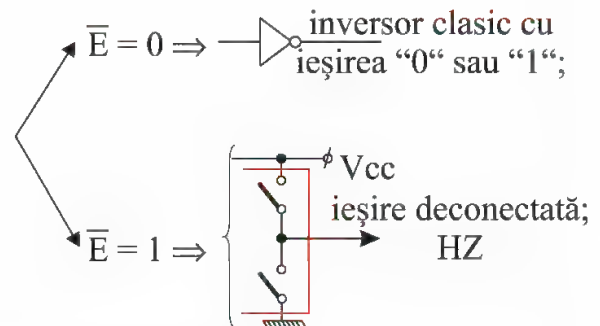
– Tabel de adevăr compact:

E	x	y
0	–	HZ
1	0	1
1	1	0

– Implementare:

**CU DECONECTARE PE “1” :**

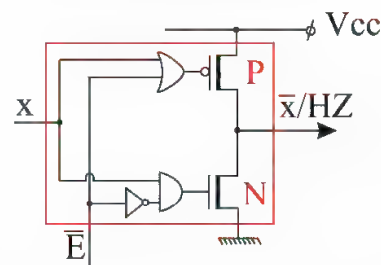
– Funcționare:



– Tabel de adevăr compact:

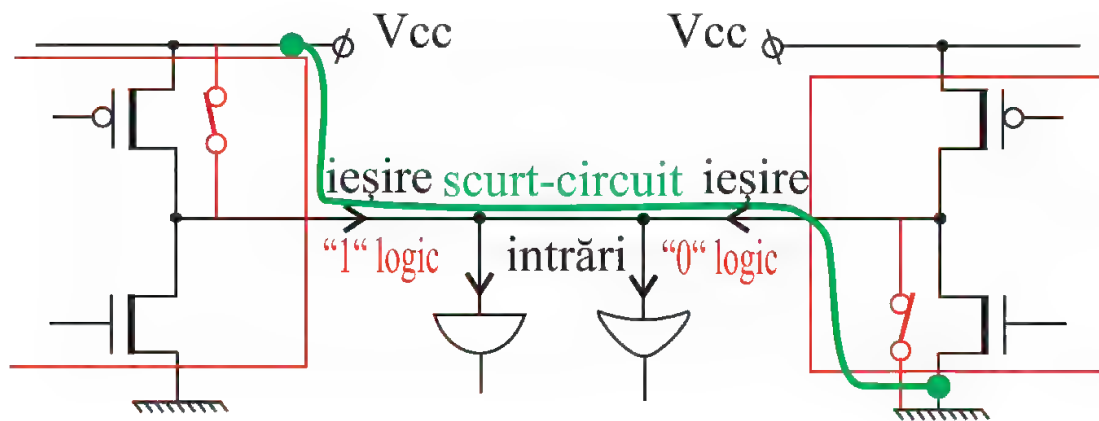
Ē	x	y
0	0	1
0	1	0
1	–	HZ

– Implementare:

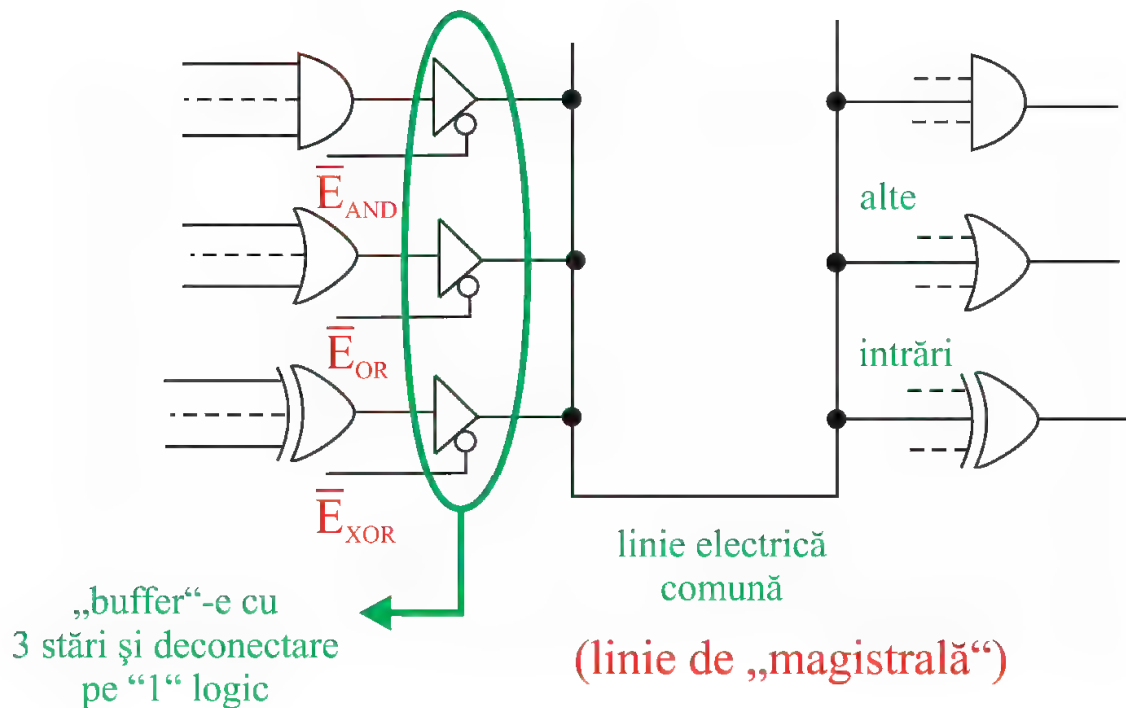
– Poarta OR se blochează la “1” $\Rightarrow$ tranzistorul P primește “1” și e deschis;– Poarta AND se blochează la “0” $\Rightarrow$ tranzistorul N primește “0” și e deschis;

$$\text{CÂND } E = 0 \Leftarrow \left(\begin{array}{c} \text{AMBELE PORTI SE} \\ \text{BLOCHEAZĂ SIMULTAN} \end{array} \right) \Rightarrow \text{CÂND } \bar{E} = 1$$

Observație: – Ieșirile unor porți obișnuite (fără HZ) **nu** se pot conecta împreună pentru că s-ar produce **scurt-circuit** când furnizează valori logice diferite.



– Pentru a conecta ieșirile mai multor circuite (porți) la aceeași linie electrică, acestea trebuie trecute **obligatoriu** prin inversoare / “buffere” cu 3 stări:



– Numai un singur circuit va avea ieșirea normală, restul fiind deconectate:

$\bar{E}_{AND}$	1	0	1	1
$\bar{E}_{OR}$	1	1	0	1
$\bar{E}_{XOR}$	1	1	1	0
Stare magistrală	complet deconectată	conectează AND	conectează OR	conectează XOR

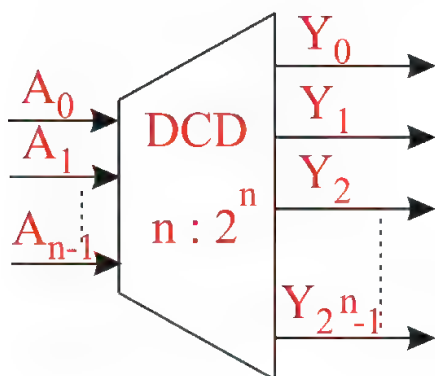
9. Câteva funcții logice particulare

– Anumite pachete de funcții logice particulare sunt des întâlnite în practică și au fost sintetizate în variantă integrată.

- Se vor considera cele mai importante:
- MULTIPLEXOARE ;
 - DEMULTIPLEXOARE ;
 - DECODIFICATOARE (DE ADRESĂ) ;
 - SUMATOARE .

9.1. Decodificatoare (de adresă)

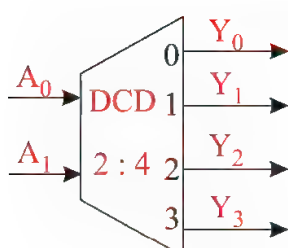
– Sunt structuri logice combinaționale cu n intrări (denumite **de adresă** sau **de selecție**) și 2^n ieșiri care funcționează astfel: dintre toate ieșirile, numai cea al cărei indice corespunde cu valoarea combinației binare aplicată pe intrări va deveni activă (pe “1” logic), pe când toate celelalte rămân inactive (pe “0” logic).



$$\begin{cases} Y_k = 1 \text{ numai pt. } k = (A_{n-1}A_{n-2} \dots A_1A_0)_2 ; \\ (k = \overline{0, 2^n - 1}); \end{cases} \quad \text{adresa ieșirii } k$$

Observație: – Fiecare ieșire implementează de fapt câte un min-term al funcției logice de cele n intrări de adresă ($Y_k = m_k$).

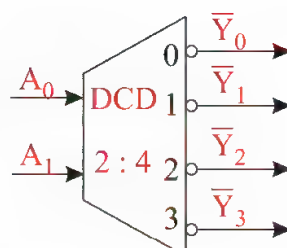
– **Exemplu:** **DCD 2:4**, cu ieșiri directe / negate:



(simbol)

A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

(tabel de adevăr)

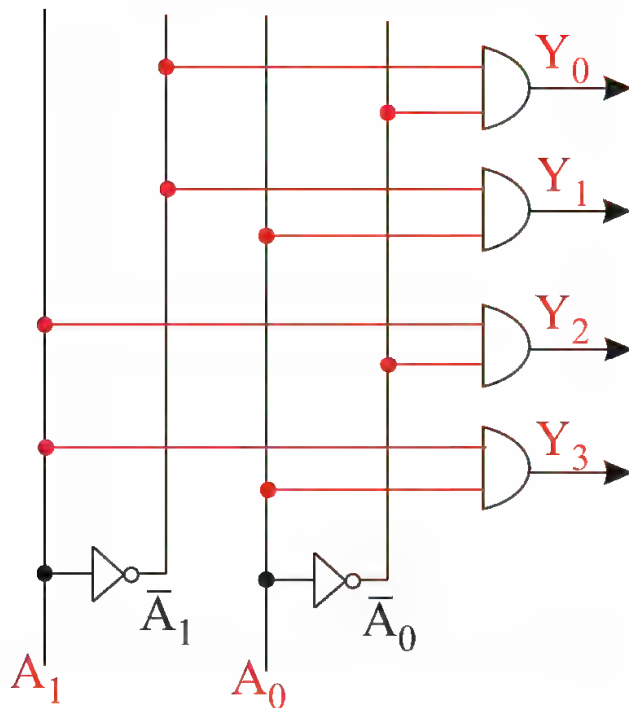


(simbol)

A_1	A_0	$\bar{Y}_3$	$\bar{Y}_2$	$\bar{Y}_1$	$\bar{Y}_0$
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

(tabel de adevăr)

– Implementare DCD 2:4 cu ieșiri directe:

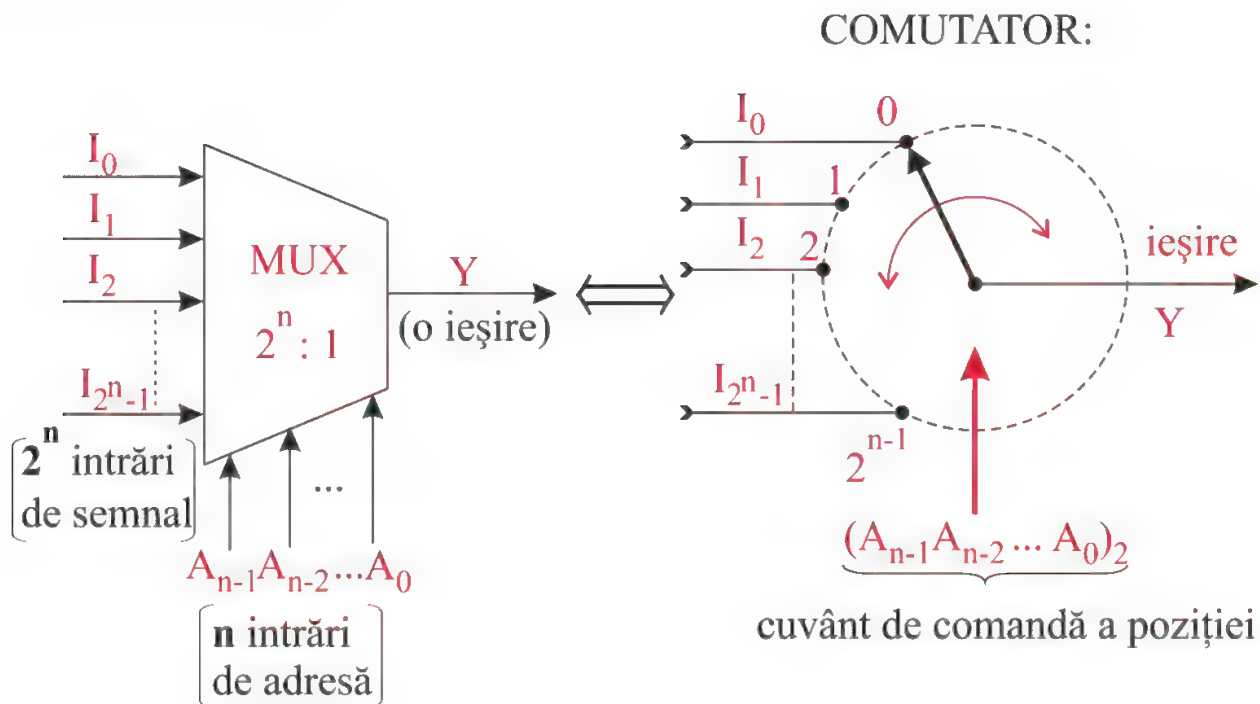


– Fiecare ieșire este o funcție AND din toate intrările, care sunt inversate sau nu conform codului binar al indicelui ieșirii.

Observatie: – Decodificatoarele de adresă sunt structuri de bază, care se utilizează în multiplexoare și demultiplexoare, memorii și selecția dispozitivelor de intrare / ieșire.

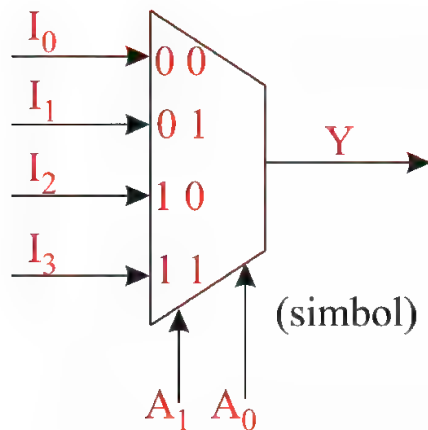
9.2. Multiplexoare

– Sunt structuri logice combinaționale echivalente unor comutatoare mecanice cu *mai multe poziții de intrare*:



– ieșirea este conectată doar la una dintre cele 2^n intrări, în funcție de adresa aplicată pe cele n intrări de selecție.

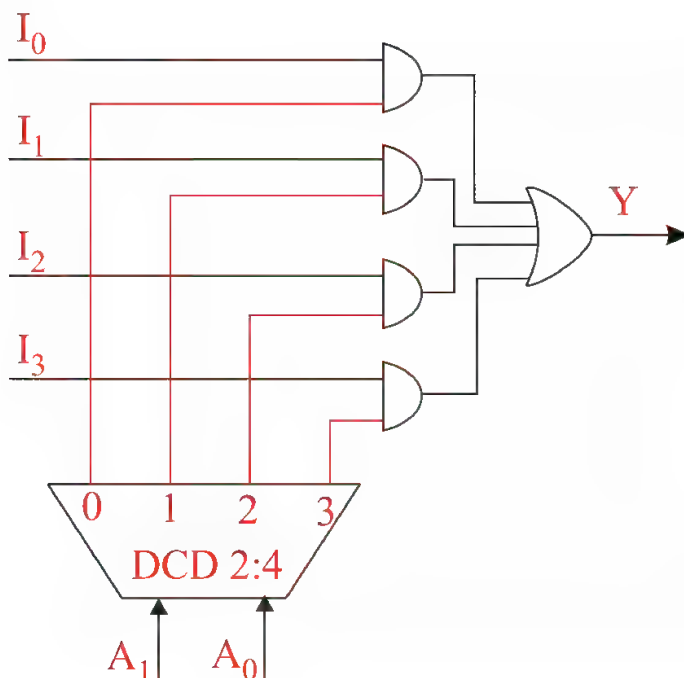
– **Exemplu: MUX 4:1 ;**



Tabel de adevăr compact

A_1	A_0	I_3	I_2	I_1	I_0	Y
0	0	-	-	-	0	$I_0 = 0$
0	1	-	-	0	-	$I_1 = 0$
1	0	-	0	-	-	$I_2 = 0$
1	1	0	-	-	-	$I_3 = 0$

– Implementare:

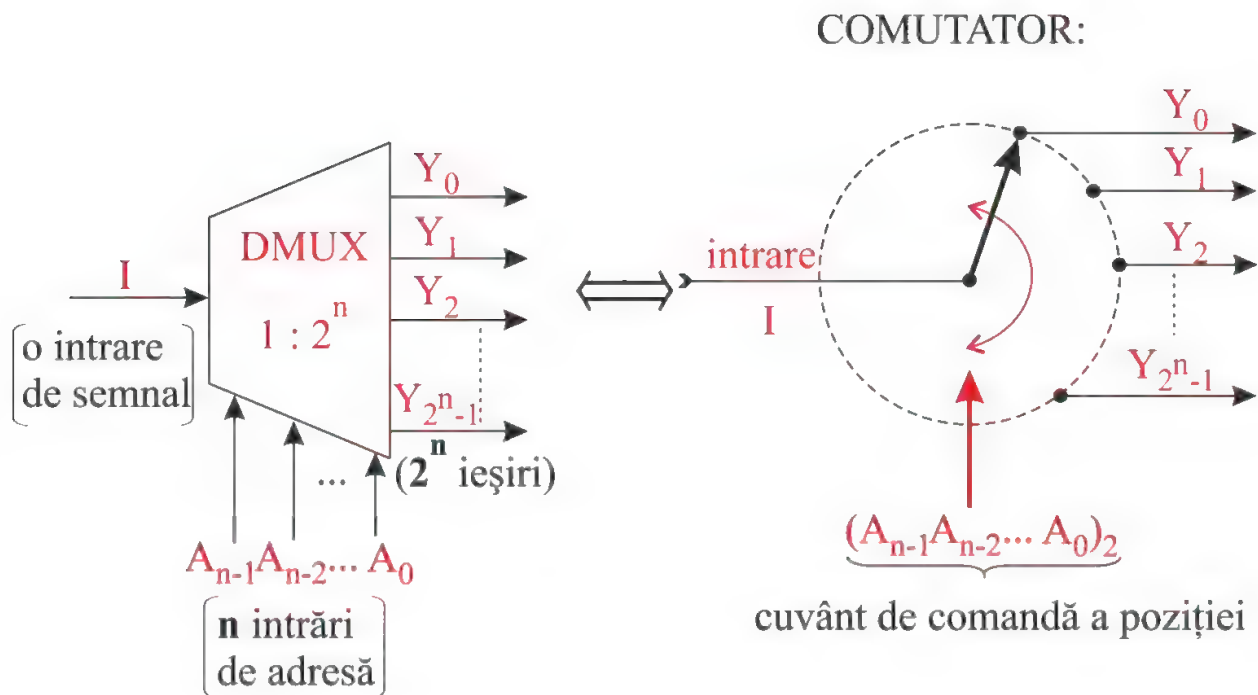


– Numai o singură poartă AND va primi “1” pe intrarea de comandă, permițând trecerea semnalului de intrare, restul fiind blocate la “0”, indiferent de valorile intrărilor;

– Practic, intrarea I_k trece prin poarta AND corespunzătoare numai când $k = (A_1 A_2)_2$, datorită decodificatorului de adrese.

9.3. Demultiplexoare

– Sunt structuri logice combinaționale cu funcționare inversă multiplexoarelor, echivalente unor comutatoare mecanice cu *mai multe poziții de ieșire*:

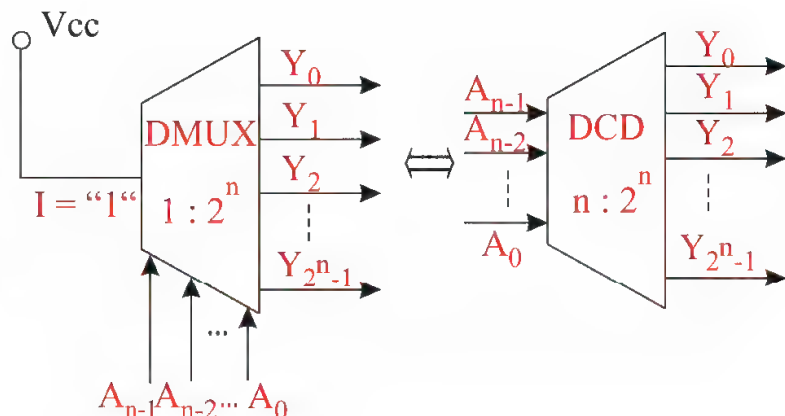


– Valoarea intrării este transmisă către o singură ieșire, aceea care este selectată prin adresa depusă pe cele n intrări de selecție.

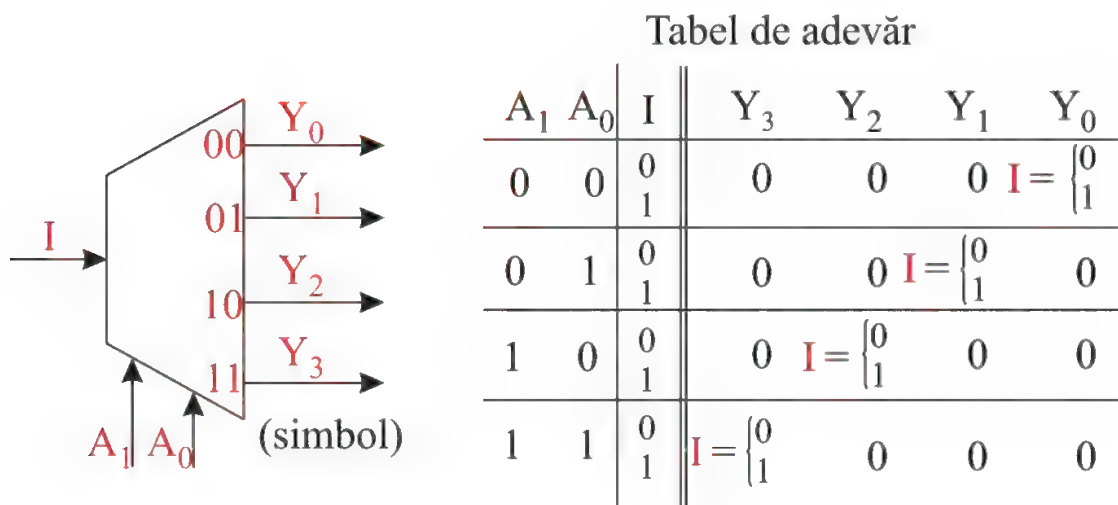
– Dacă la multiplexor echivalența cu comutatorul mecanic este completă, în cazul demultiplexorului există totuși o diferență semnificativă față de comutatorul mecanic: la structura logică ieșirile nefolosite capătă valoarea "0" logic (ieșirile la care nu se transmite semnalul de intrare), dar la comutatorul mecanic ele rămân complet deconectate, adică în starea de înaltă impedanță (HZ).

Observație:

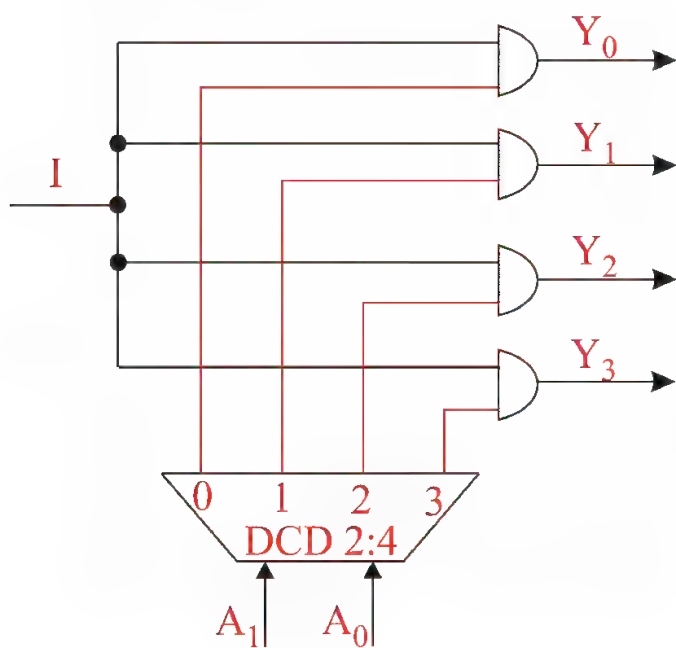
– Dacă intrarea este permanent conectată la "1" logic, demultiplexorul devine decodificator de adresă:



– Exemplu: **DMUX 1:4**;



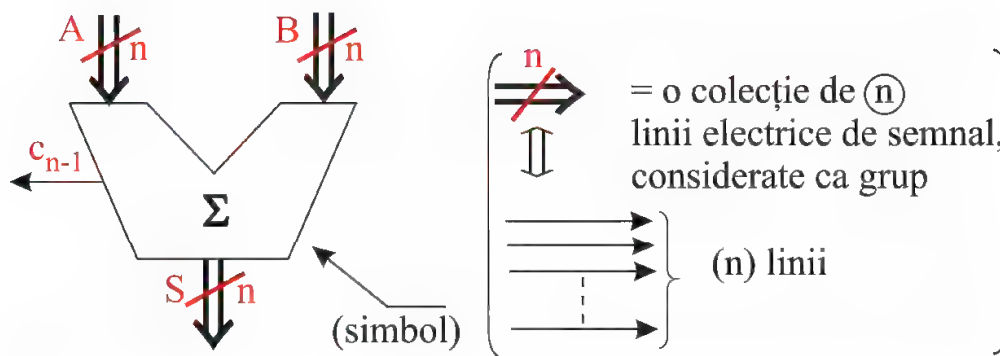
– Implementare:



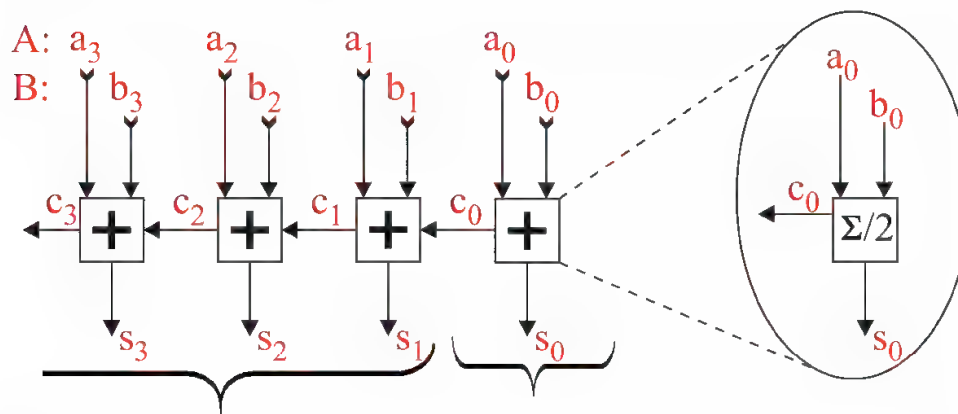
– Se bazează tot pe utilizarea unui decodor care deschide o singură poartă AND cu “1” logic pe intrarea de comandă, permițând trecerea intrării numai la acea ieșire Y_k care are codul binar $k = (A_1A_0)_2$.

9.4. Sumatoare

– Sunt structuri logice combinaționale care implementează calculul sumei a două numere binare, întregi și fără semn.



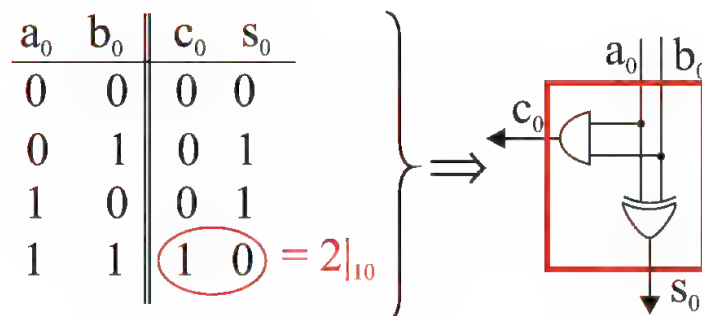
– **Exemplu:** – Sumator pe 4 biți;



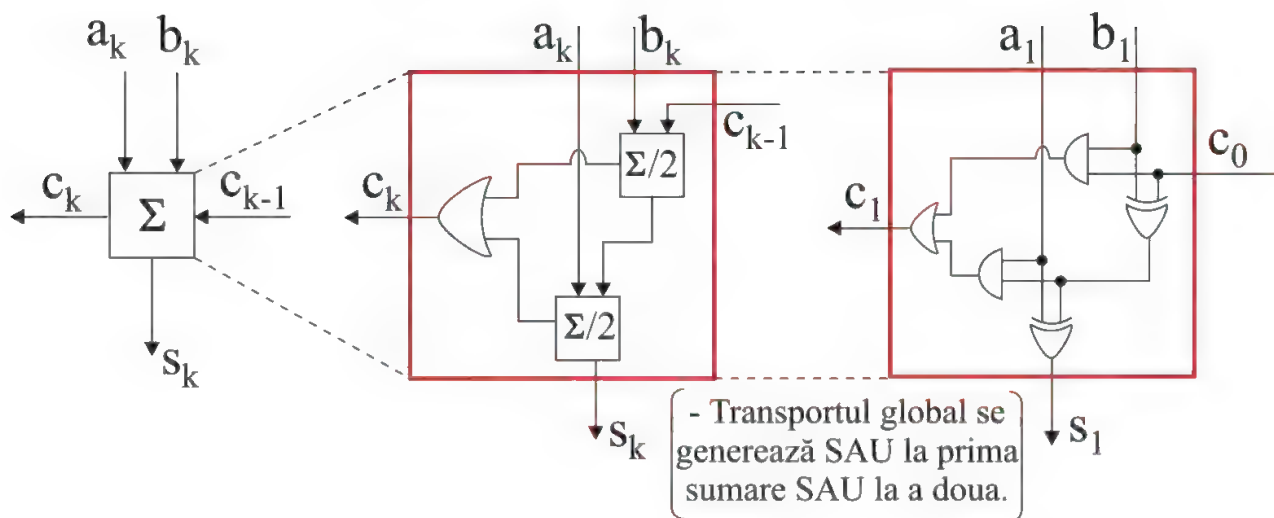
sumatoare complete de un rang (sumează câte 3 biți)

semi-sumator (sumează câte 2 biți)

– Implementarea semi-sumatorului:

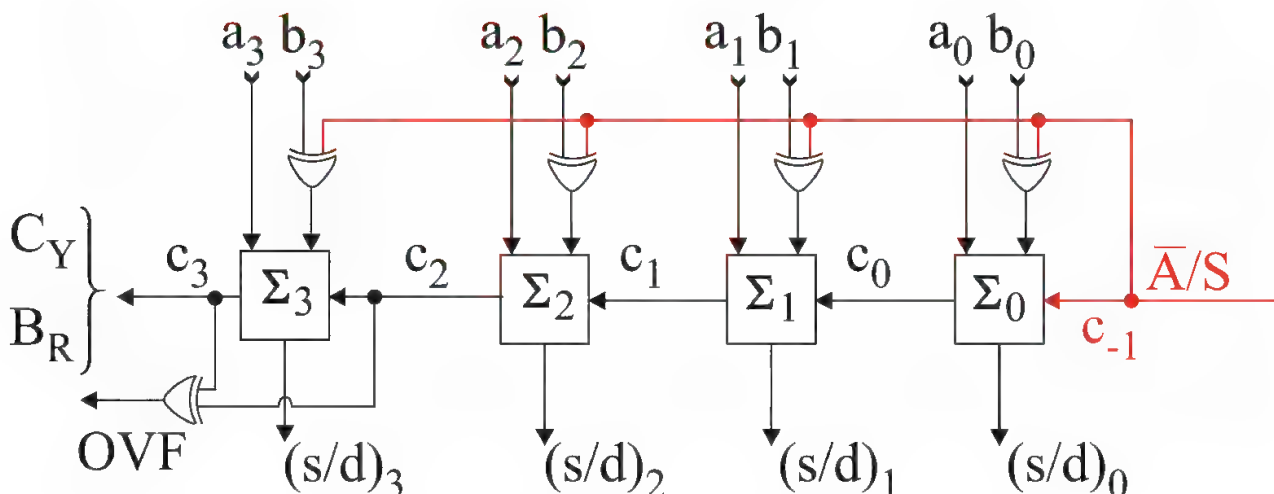


– Implementarea sumatorului complet de un rang se poate face considerând suma primilor doi biți sumată ca rezultat cu cel de-al treilea bit (asociativitatea sumei):



Observatie: – Scăderea se poate implementa printr-o adunare cu complementul față de 2 al scăzătorului (număr negativ);

– Implementarea unui circuit aritmetic (sumator / scăzător):



– Porțile XOR sunt folosite ca inversoare comandate:

- lasă nemodificați biții numărului B când $\bar{A}/S = 0 \Rightarrow +B$;
- inversează biții numărului B când $\bar{A}/S = 1 \Rightarrow -(B)_{c_1}$;

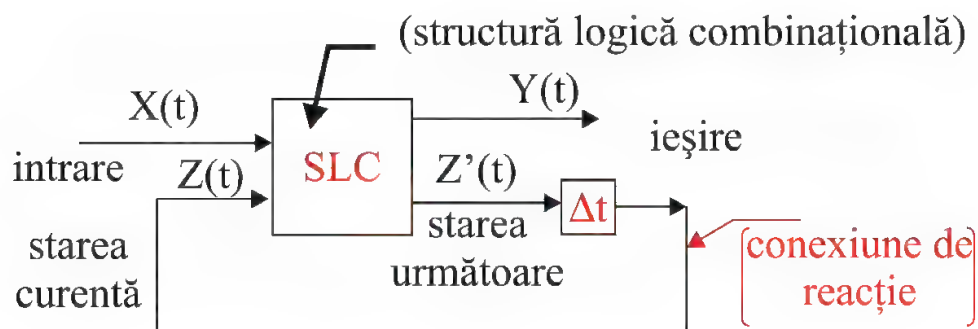
– Sumatorul Σ_0 este complet și are rolul de a mai aduna un bit $\boxed{c_{-1} = 1}$ când

$$\overline{A}/S = 1 \Rightarrow (-B)_{c_1} \rightarrow (-B)_{c_2} ;$$


$$\begin{aligned} & \overline{A}/S = 0 \mid \Rightarrow ADD \text{ (adunare): } S = A + B ; \\ & \overline{A}/S = 1 \mid \Rightarrow SUBTRACT \text{ (scădere): } D = A + (-B)_{c_2} . \end{aligned}$$

10. Structuri logice secvențiale

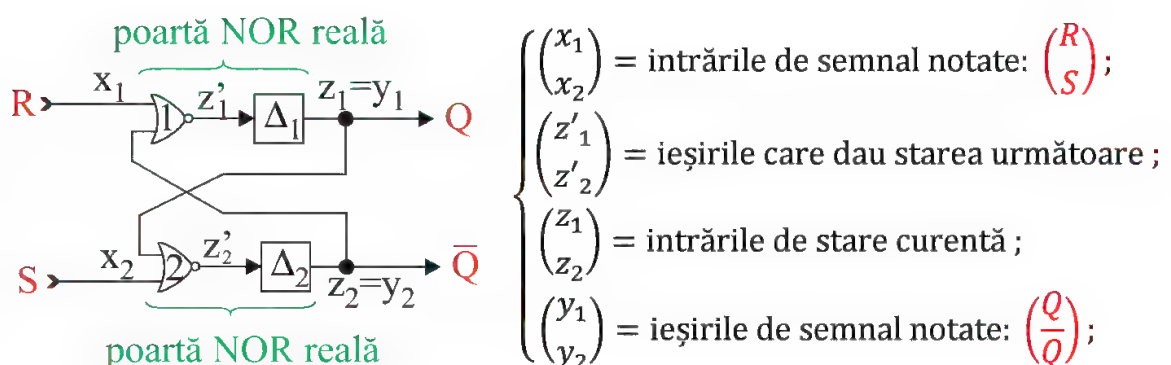
- Sunt caracterizate prin prezența a cel puțin unei conexiuni **de reacție** de la ieșire la intrarea structurii;
- Datorită conexiunii de reacție, la analiza lor trebuie introdusă și variabila **timp** $\Rightarrow$ structura se comportă diferit de la un moment de timp la altul:



- Starea curentă va deveni starea următoare după întârzierea Δt : $Z(t + \Delta t) = Z'(t)$.

10.1. Structuri secvențiale cu memorie (celula RS)

- Sunt structuri logice secvențiale cu două stări stabile, care au la bază **celula RS**, formată din 2 porți NOR între care există reacții încrucișate prin intermediul întârzierilor Δ_1 și Δ_2 (adică chiar timpii de propagare prin porțile REALE, la care ieșirea nu se modifică instantaneu):



Observații: – Cuvântul de ieșire pe 2 biți $\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$, adică este identic cu starea prezentă;

– Din cauza **reacției în buclă sub formă de “8”** dintre cele două porți (o intrare a porții 1 depinde de ieșirea porții 2, care are una din intrări ce depinde la rândul ei de ieșirea porții 1), analiza funcționării schemei nu se poate face decât dacă se întrerupe această reacție;

– Întreruperea buclei la una din porți se poate face prin plasarea pe intrarea care nu face parte din reacție a unei valori logice “0” sau “1”, în funcție de tipul porții, astfel încât **ieșirea ei să fie indiferentă la valoarea intrării de reacție**. Intrarea care nu face parte din reacție este considerată ca și cum ar fi o intrare de comandă ce “închide” poarta la o valoare fixată și nu lasă să treacă prin ea semnalul de reacție de la cealaltă intrare.

– **Funcționare:** – Pentru analiza funcționării celulei, trebuie parcurse cel puțin 5 etape, în ordinea următoare:

(1) – Se întrerupe reacția doar în punctul 1, nu și în punctul 2 (poarta 1 “închisă”, poarta 2 “transparentă”) și se verifică ieșirile rezultate;

(2) – Se lasă neîntreruptă reacția în ambele puncte (ambele porți “transparente”), pentru a verifica efectul buclei asupra stării structurii logice;

(3) – Se întrerupe reacția doar în punctul 2, nu și în punctul 1 (poarta 2 “închisă”, poarta 1 “transparentă”) și se verifică noua stare rezultată prin citirea ieșirilor;

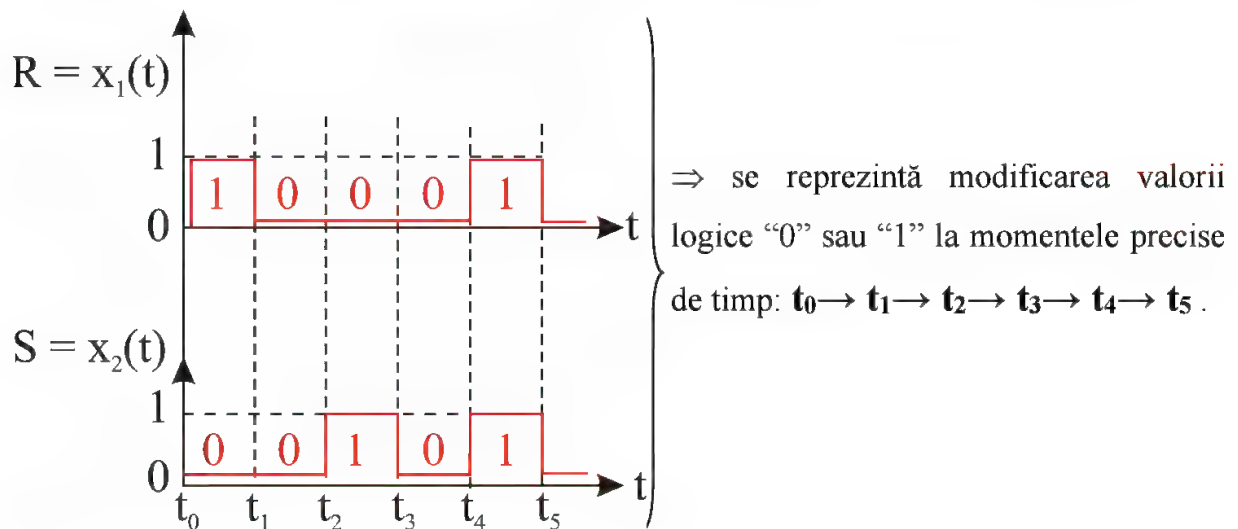
(4) – Se verifică din nou efectul buclei, lăsând reacția să se desfășoare neîntreruptă în ambele puncte (ambele porți “transparente”);

(5) – Se întrerupe reacția în ambele puncte (ambele porți “închise”), separând astfel structura în două părți independente fără legătură între ele; în această situație nu va rezulta o stare utilă (cele două porți se comportă independent, ca și cum nu ar mai fi interconectate).

Urmărind aceste etape, în cazul particular al celulei formată din 2 porți NOR va rezulta următoarea evoluție în timp a cuvântului de intrare:

$$\begin{pmatrix} R \\ S \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

– Această evoluție se poate reprezenta mai comod folosind o **diagramă de timp**:

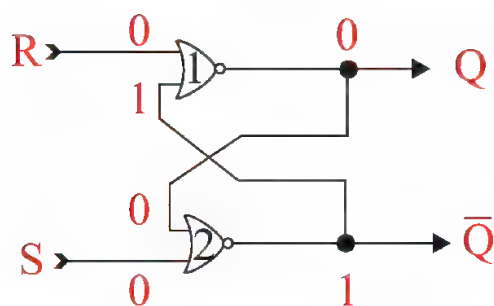


$$t_0: \begin{pmatrix} R \\ S \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\boxed{R = 1}$$

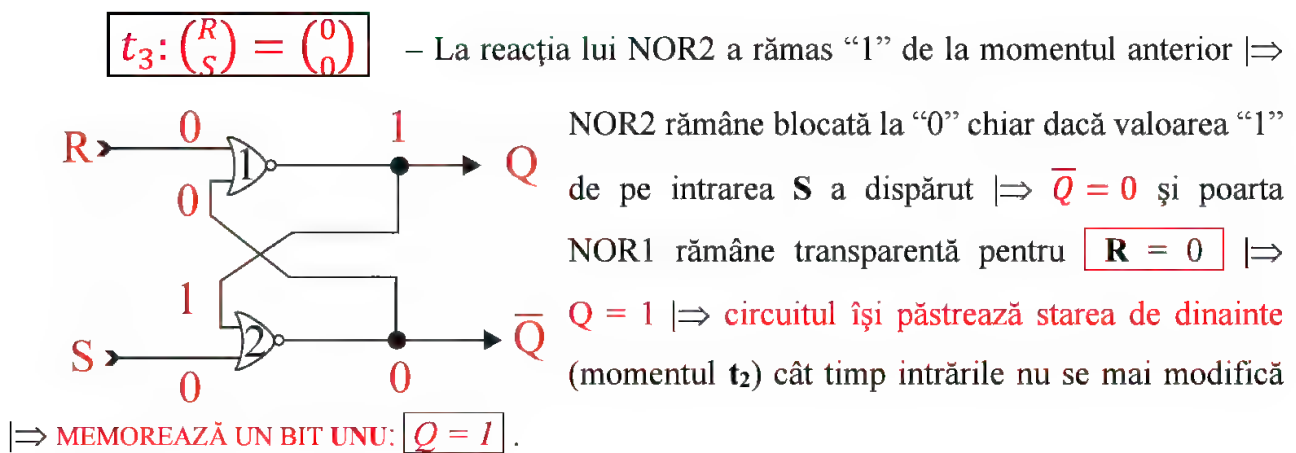
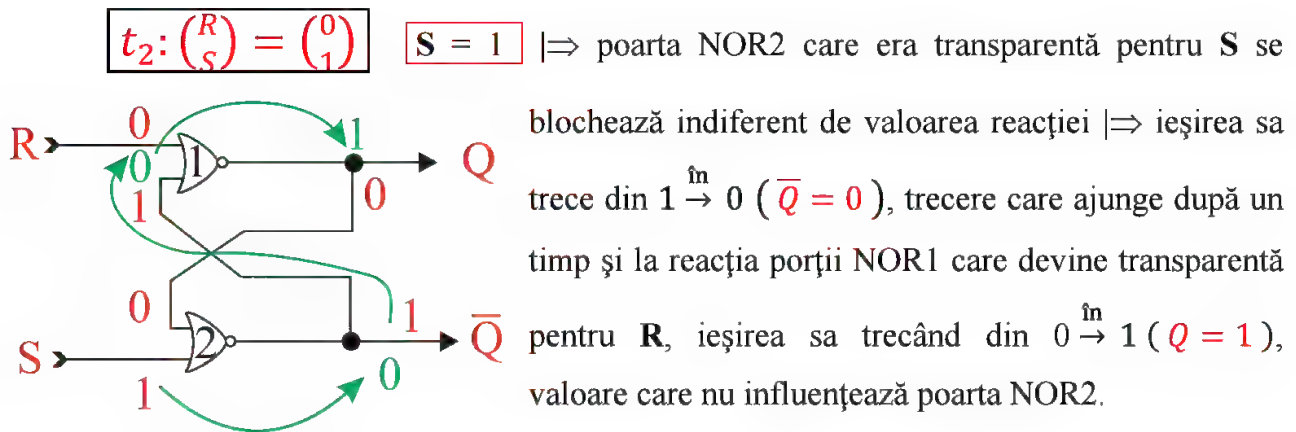
$\Rightarrow$ poarta NOR1 se blochează la “0” indiferent de valoarea intrării de reacție $\Rightarrow$ după o întârziere Δ_1 modificarea ajunge la ieșire: ($Q = 0$) și la reacția porții NOR2 care devine transparentă pentru $\boxed{S = 0}$ $\Rightarrow$ după o întârziere Δ_2 la ieșirea ($\bar{Q} = 1$) și la intrarea de reacție a porții NOR1 ajunge “1” logic.

$$t_1: \begin{pmatrix} R \\ S \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$



– La reacția lui NOR1 a rămas “1” de la momentul anterior $\Rightarrow$ poarta NOR1 rămâne blocată la “0” chiar dacă intrarea R s-a modificat din 1 $\xrightarrow{\text{în}}$ 0 $\Rightarrow Q = 0$ și reacția lui NOR2 rămâne la “0” și cum $\boxed{S = 0}$ $\Rightarrow \bar{Q} = 1$ $\Rightarrow$ **circuitul își păstrează starea de dinainte** (momentul t_0) cât timp intrările nu se mai modifică

$\Rightarrow$ **MEMOREAZĂ UN BIT ZERO: $\boxed{Q = 0}$** .



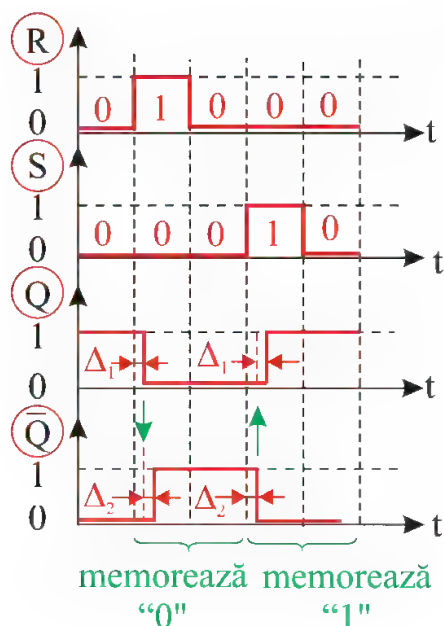
$$t_4: \begin{pmatrix} R \\ S \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \left\{ \begin{array}{l} \boxed{R = 1} \Rightarrow \text{poarta NOR1 se blochează la "0"} \quad (Q = 0) \\ \boxed{S = 1} \Rightarrow \text{poarta NOR2 se blochează la "0"} \quad (\bar{Q} = 0) \end{array} \right. \Rightarrow$$

ambele ieșiri devin “0” și nu mai sunt una negatul celeilalte $\Rightarrow$ **stare inutilă NEDORITĂ** $\Rightarrow$ se evită activarea simultană a intrărilor R și S pe “1” logic.

Observație: – Celula **RS** este o celulă elementară de memorie și poate memora un bit de informație, dar trebuie comandată prin 2 semnale diferite care nu pot fi active simultan:

$$\begin{cases} \mathbf{R = "RESET"} & \Rightarrow \text{aduce } \mathbf{Q} \text{ la } \mathbf{0}; \\ \mathbf{S = "SET"} & \Rightarrow \text{aduce } \mathbf{Q} \text{ la } \mathbf{1}; \end{cases}$$

– Evoluția ieșirilor în funcție de intrări se poate reprezenta prin următoarea diagramă de timp:

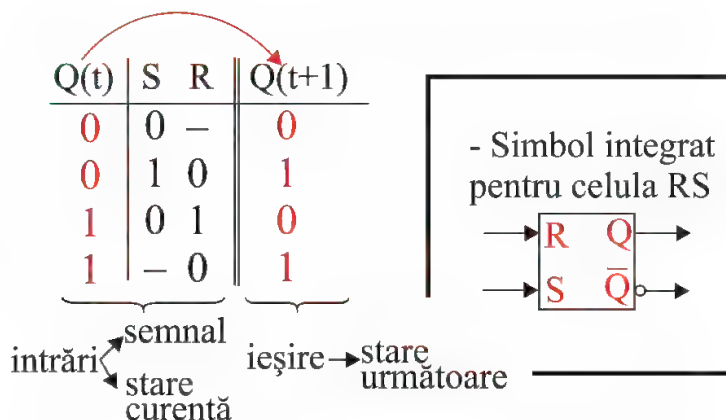


- sau printr-un tabel de adevăr:

R	S	$Q(t+1)$ = starea următoare
0	0	$Q(t)$ = starea prezentă
0	1	1 = set
1	0	0 = reset
1	1	– = stare invalidă

$$[Q(t+1) = Q(t+\Delta_1+\Delta_2)]$$

– Tabelul de adevăr se poate modifica în așa fel încât să indice condițiile de trecere de la starea prezentă la starea următoare
 $\Rightarrow$ **tabel de excitație** :



11. Perfecționarea celulei RS

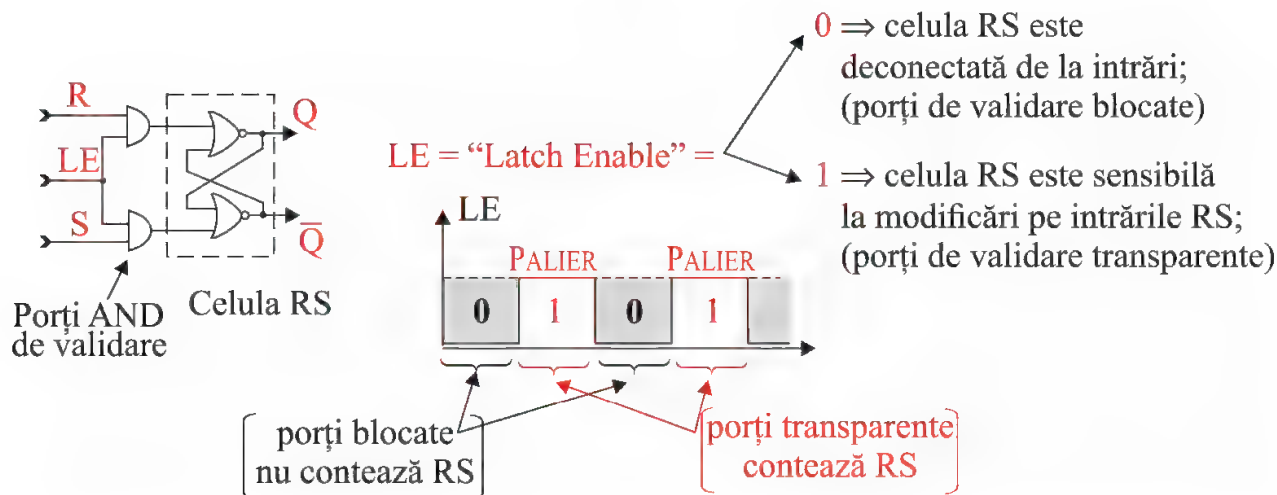
– Celula RS se poate îmbunătăți prin:

- eliminarea stării instabile nepermise ;
- comanda simultană a celor două intrări ;
- determinarea precisă a momentului modificării .

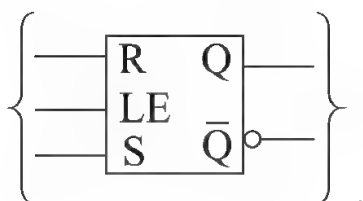
11.1. Precizarea momentului de comandă

11.1.1. Celula RS de tip “LATCH” (zăvor):

– Se introduce un semnal de intrare suplimentar care să valideze configurația intrărilor de comandă R și S numai în anumite intervale de timp:



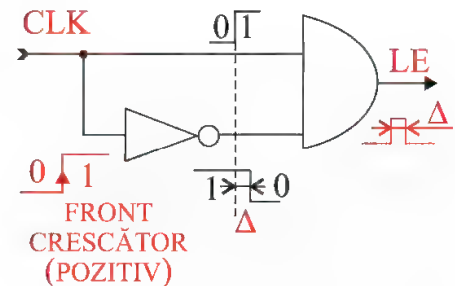
– Celula RS cu validare pe PALIER se numește **LATCH-RS** și are simbolul:



11.1.2. Bistabilul RS sincron pe front:

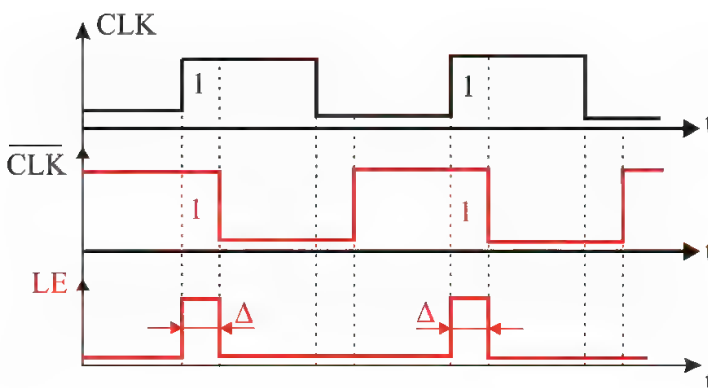
– Se dorește micșorarea cât mai mult posibilă a intervalului de timp când celula RS este sensibilă la modificările intrărilor, pentru a preciza cât mai exact momentul de comutare.

– Se transformă **frontul pozitiv** al semnalului de validare într-un impuls foarte îngust și constant:



– Semnalul de validare este trecut printr-o poartă AND, la care ajunge pe două căi cu **întârzieri diferite**:

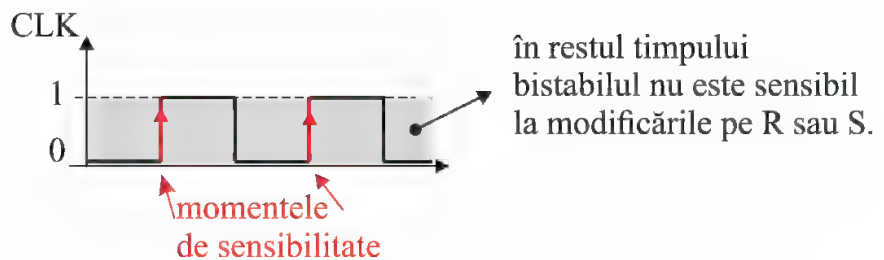
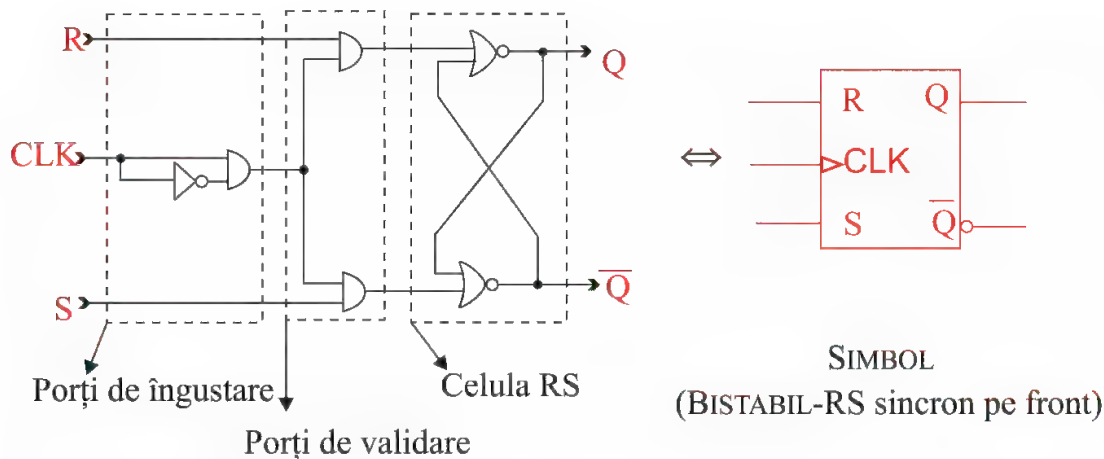
- direct la intrarea de sus ;
- inversat, după un timp Δ la intrarea de jos ;



– Poarta AND va produce un semnal $LE = "1"$ la ieșire numai pe durata foarte scurtă și constantă Δ , când ambele sale intrări sunt "1" logic.

Observație: – Dacă timpul de propagare prin toate tipurile de porți este identic și egal cu $\Delta \Rightarrow$ celula **RS** va avea timp să "vadă" modificarea pe una din intrările **R** sau **S** până la ieșire, dar nu suficient cât să comute de mai multe ori pentru că timpul de propagare prin bucla de reacție este 2Δ .

– În acest fel s-a obținut o celulă **RS** sensibilă numai în momentul frontului crescător al semnalului de validare, denumit aici "**CLOCK**" (= ceas):



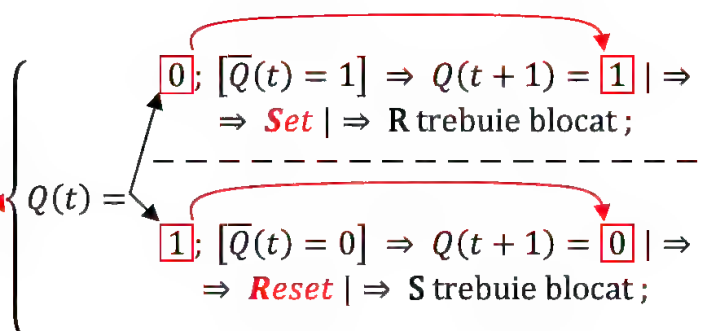
11.2. Eliminarea nedeterminării: bistabilul JK

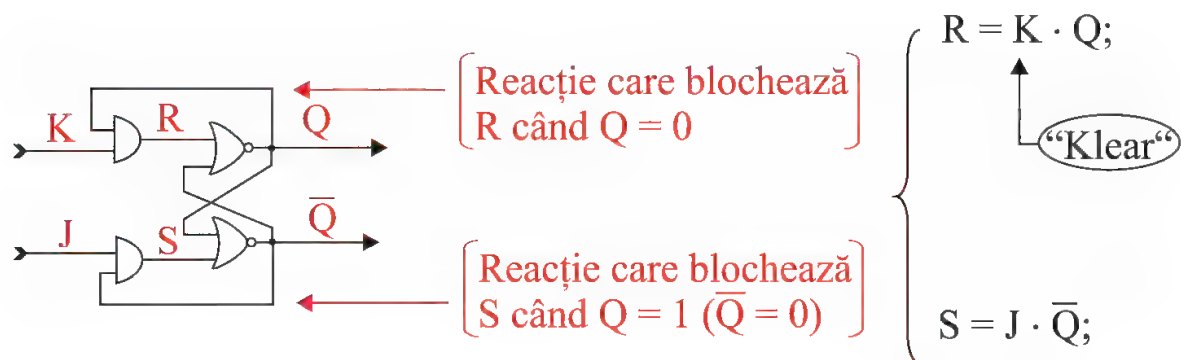
– Pentru eliminarea nedeterminării în evoluția bistabilului **RS** sincron pe front care apare în momentul când ambele intrări sunt active la frontul pozitiv al ceasului (**R = S = "1"**), semnalele de comandă trebuie **alterate** înainte de a ajunge la celula elementară **RS**.

– Ar fi preferabil ca celula RS să comute în starea **complementară** când ambele comenzi sunt active:

TABEL DE ADEVĂR RS→JK

R S	Q(t+1)	Q(t+1) modificat
0 0	Q(t)	Q(t)
0 1	1	1
1 0	0	0
1 1	INVALID	$\bar{Q}(t)$





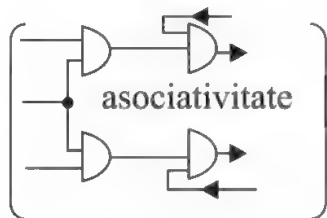
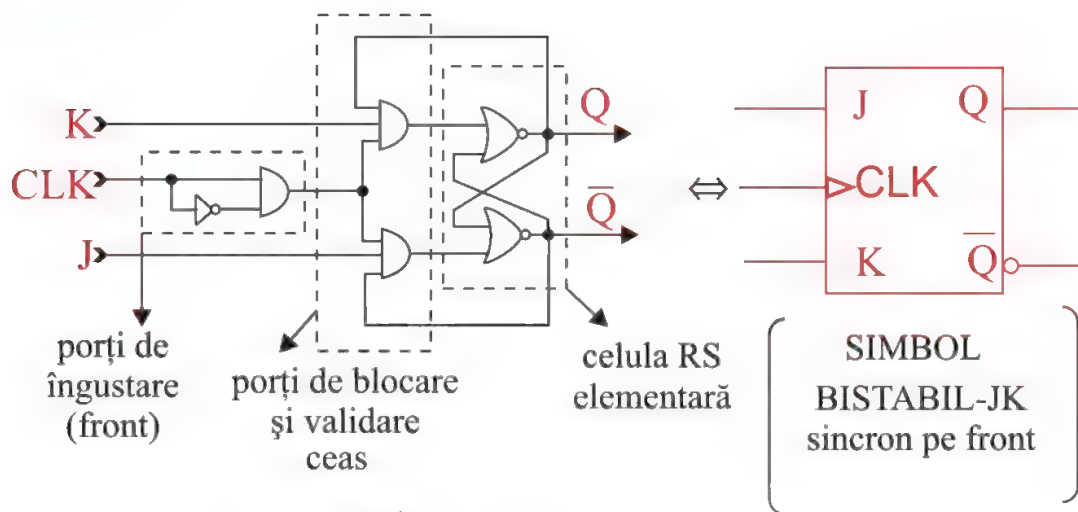
– Când noile intrări sunt $\mathbf{JK} = “11”$, se transmite celulei una din cele două combinații permise, în funcție de starea curentă:

$$RS = \begin{cases} \mathbf{0\ 1} & \text{când } Q = 0 ; \\ \mathbf{1\ 0} & \text{când } Q = 1 . \end{cases}$$

Observații:

– Blocarea lui **R** când $Q = 0$ nu afectează linia de **“Reset”** din tabelul de adevăr care ar aduce Q la 0 ;

– Blocarea lui **S** când $Q = 1$ nu afectează linia de **“Set”** din tabelul de adevăr care ar aduce Q la 1 ;



– Tabel de excitație

$Q(t)$	J	K	$Q(t+1)$
0	0	–	0
0	1	–	1
1	–	1	0
1	–	0	1

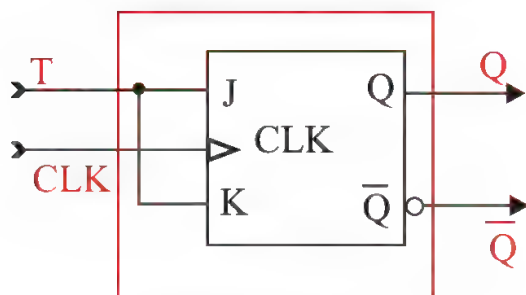
are 2 valori indiferente în plus față de tabelul de excitație RS

– Bistabilul **JK** poate fi construit numai în varianta **sincronă pe front** pentru că în varianta “latch” ar avea timp să comute de mai multe ori în starea complementară și înapoi pe durata palierului dacă ambele intrări **JK** = “11” !

11.3. Comanda simultană a intrărilor: bistabilii *T* și *D*

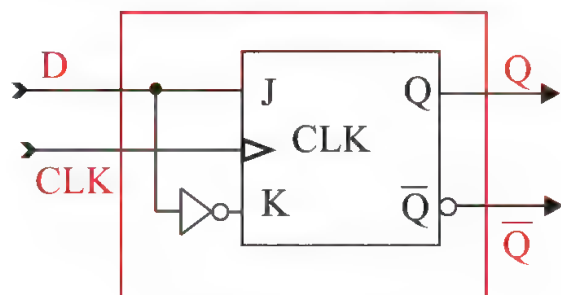
– Comanda simultană a celor două intrări de comandă **J** și **K** se poate realiza în două moduri:

INTRĂRI IDENTICE:



BISTABIL T
sincron pe front
(T = “TOGGLE”)

INTRĂRI COMPLEMENTARE:



BISTABIL D
sincron pe front
(D = “DATA”)

– Tabel de adevăr T:

T	Q(t+1)
0	Q(t)
1	Q̄(t)

[păstrează doar liniile
din capetele tabelului
de adevăr JK]

– Tabel de adevăr D:

D	Q(t+1)
0	0
1	1

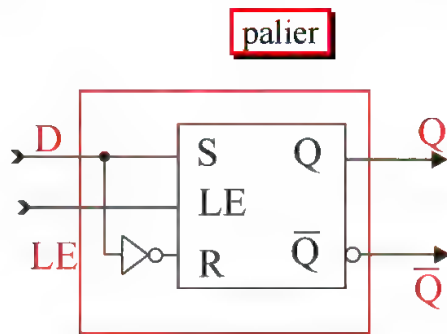
[păstrează doar liniile
din mijlocul tabelului
de adevăr JK]

Observații: – Bistabilul **T** comută în starea complementară la fiecare front crescător al ceasului CLK dacă **T** = 1 sau își păstrează starea dacă **T** = 0 ;

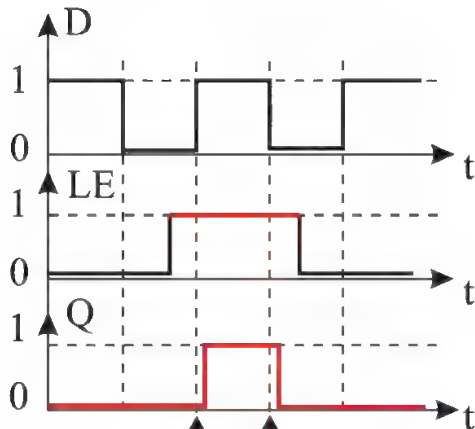
– Bistabilul **D** copie valoarea de la intrare în momentul frontului crescător al ceasului CLK $\Rightarrow$ **memorează valoarea bitului D** $\Rightarrow$ lucrează ca o **CELULĂ DE MEMORIE** cu înscriere la frontul crescător al ceasului !

11.4. Eliminarea nedeterminării și comanda simultană:**“Latch”-ul de tip D**

– Dacă intrările **R** și **S** ale celulei **RS** sunt complementare, ele se pot comanda simultan și se elimină și nedeterminarea, pentru ambele tipuri de sincronism:



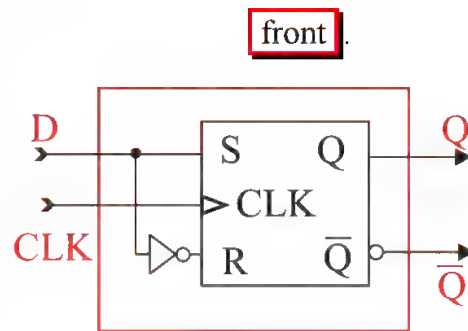
LATCH D
transparent pe palier
(obținut dintr-un “latch”
RS validat pe palier)



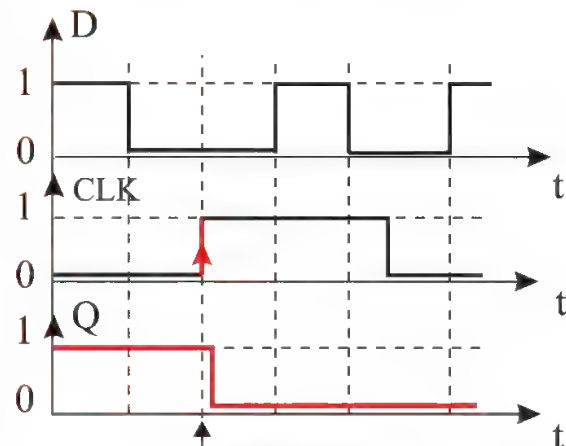
(aici “latch”-ul sesizează modificarea datelor pe D)

- Orice modificare pe intrarea D se transmite la ieșire când **LE=1** ⇒ “latch”-ul este transparent pe palier;
- Numai când **LE=0** ⇒ “latch”-ul este închis și memorează ultima valoare a lui D.

și



BISTABIL D
sincron pe front
(obținut direct dintr-un bistabil
RS comandat prin front)



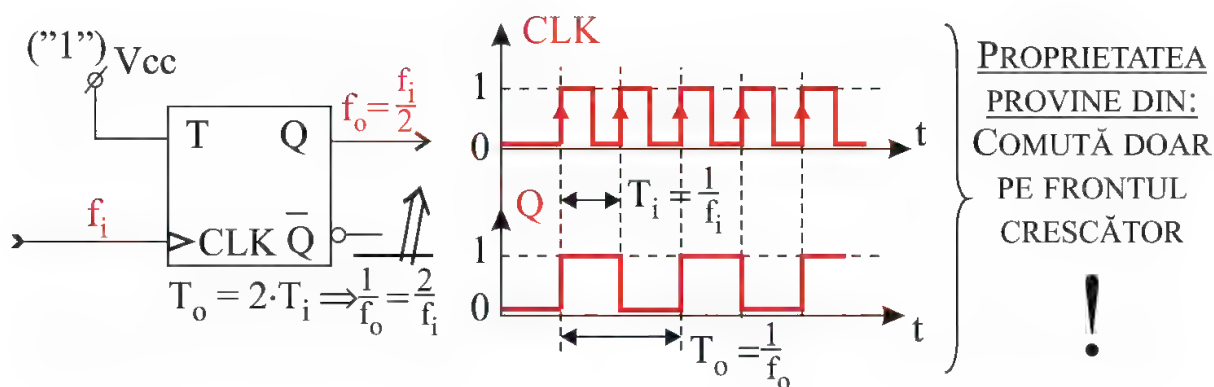
(aici bistabilul se încarcă cu valoarea datelor din momentul frontului)

- Modificarea datelor pe intrarea D nu se transmite niciodată direct la ieșire ⇒ bistabilul nu este transparent, pentru că preia doar valoarea din momentul frontului.

12. Aplicații directe ale bistabilelor

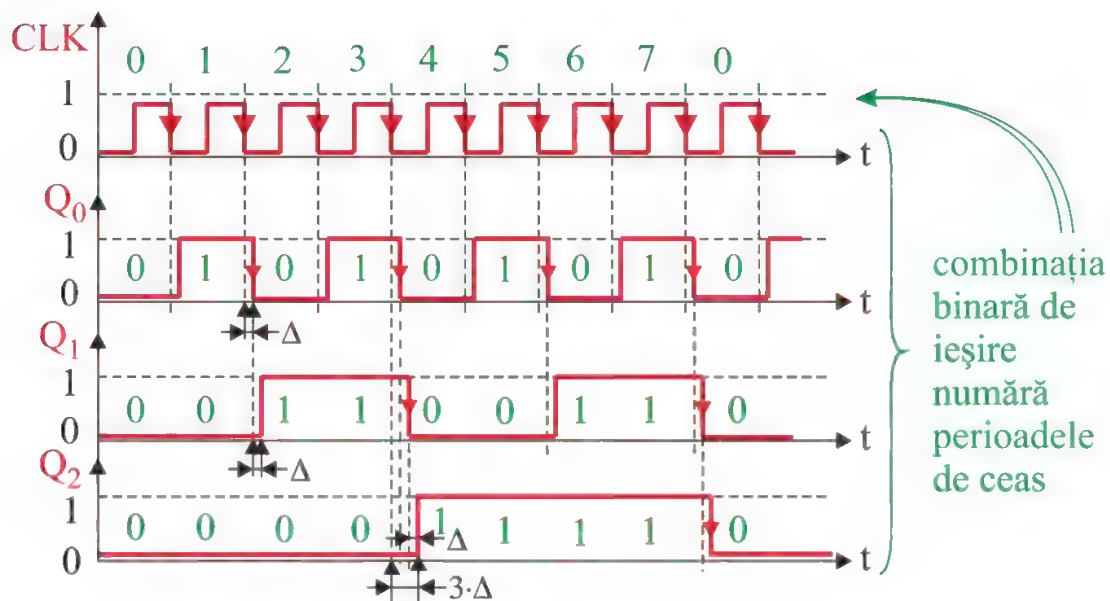
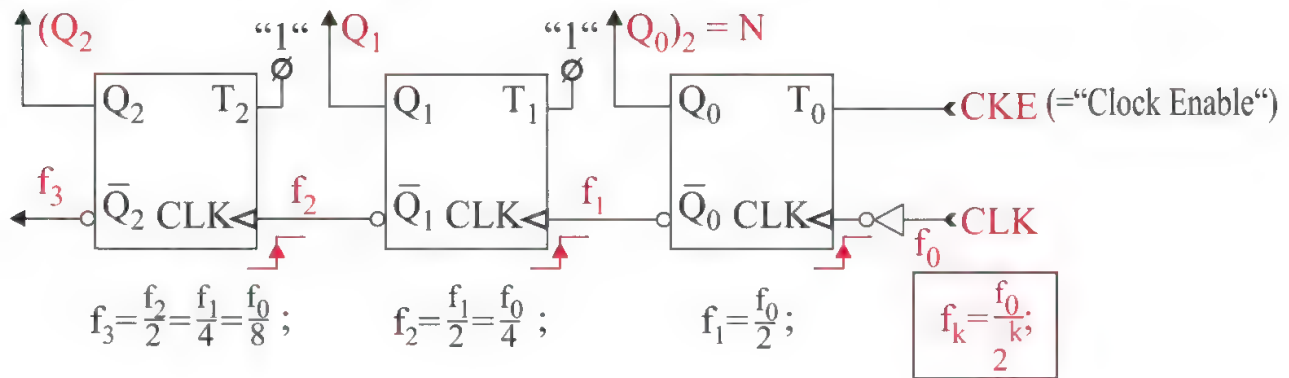
12.1. Numărătoare

- Sunt structuri secvențiale care numără impulsurile aplicate la intrarea de ceas.
- Se construiesc pe baza proprietății bistabilului de tip T de a diviza cu 2 frecvența semnalului aplicat la intrarea de ceas atât timp cât $T = 1$;



12.1.1. Numărătoare asincrone:

– Se caracterizează prin faptul că nu toate bistabilele comută în același moment, ci după încă o întârziere Δ față de cel anterior:



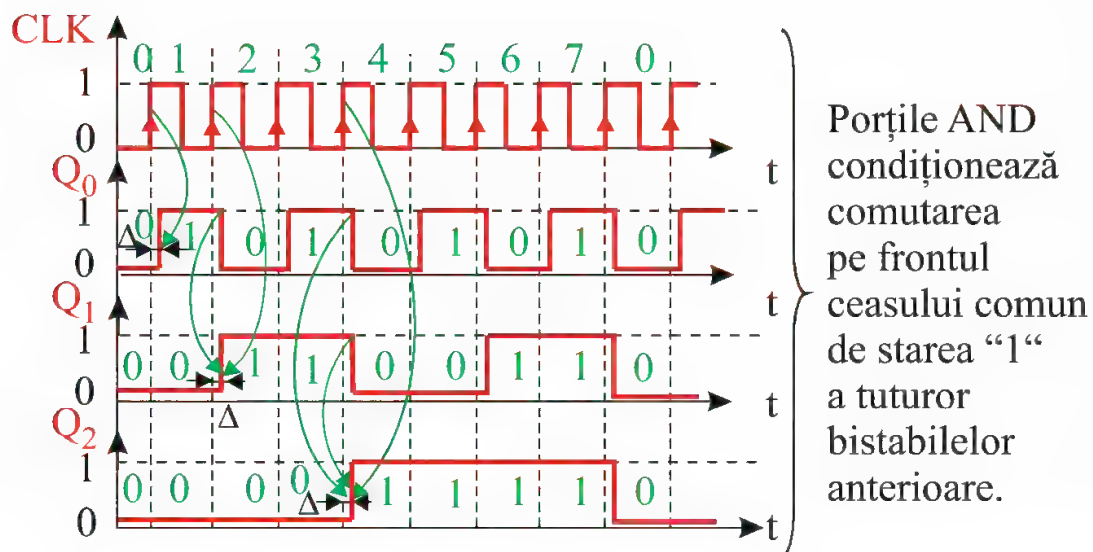
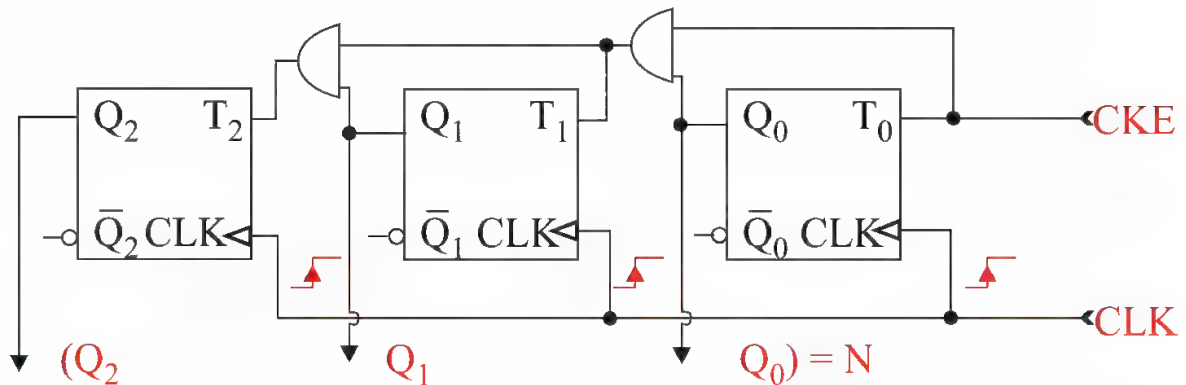
– Fiecare bistabil comută când intrarea sa de ceas trece din $\boxed{0 \uparrow 1}$, dar cum semnalul de ceas este conectat la ieșirea negativă a bistabilului anterior $\Rightarrow$ va comuta de fapt la trecerea din $\boxed{1 \downarrow 0}$ a bistabilului T anterior.

– Ansamblul se incrementează cu o unitate la fiecare front căzător al ceasului de intrare, care este și el inversat, dar numai dacă $\boxed{\text{CKE} = "1"}$; altfel își păstrează starea.

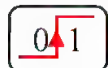
– Fiecare bistabil comută cu o întârziere Δ de la modificarea intrării de ceas $\Rightarrow$ până comut tot ansamblul trec $\boxed{3 \times \Delta}$ de la comutarea ceasului CLK.

12.1.2. Numărătoare sincrone:

- Se caracterizează prin faptul că toate bistabilele comută simultan pe același front al ceasului de intrare.
- Se construiesc prin conectarea în comun a tuturor intrărilor de ceas ale bistabilelor și folosirea intrărilor T pentru validarea comutării, în funcție de starea bistabilelor anterioare.



- Fiecare bistabil comută pe frontul crescător al ceasului comun de intrare dar numai dacă intrarea de control $T = "1"$, altfel își păstrează starea.



- Frontul crescător al ceasului va fi luat în considerare numai dacă ieșirea bistabilului anterior era "1" și comanda sa T era "1", adică ieșirea tuturor bistabilelor anterioare era "1" în momentul apariției frontului.

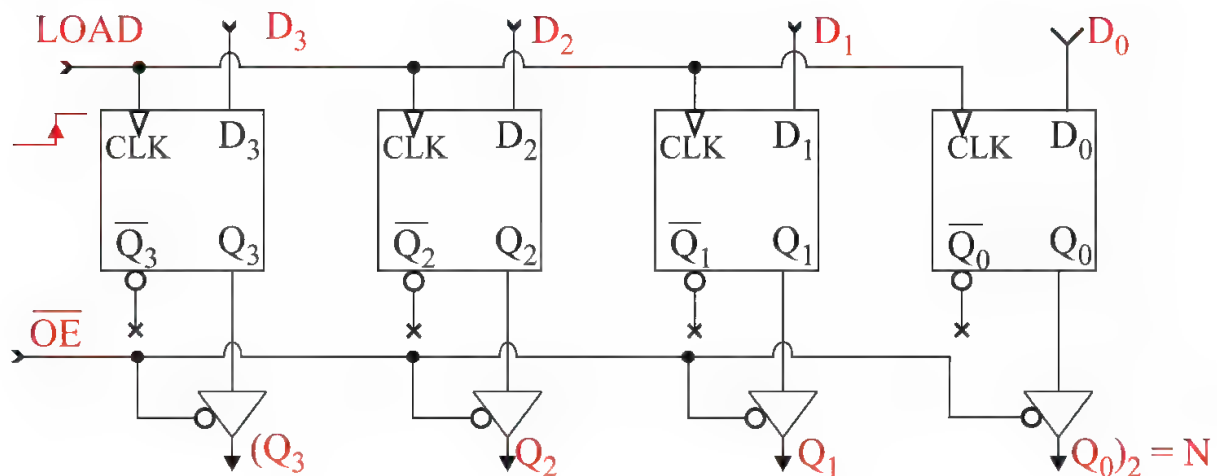
– Ansamblul numără perioadele de ceas în sens crescător, incrementându-se după ce trece un timp Δ , dacă intrarea de validare este: $\boxed{\text{CKE} = "1"}$.

12.2. Registre

- Sunt structuri secvențiale care memorează valoarea unui număr binar pe $\boxed{n}$ biți.
- Se construiesc pe baza proprietății bistabilului de tip **D** de a memora valoarea aplicată la intrarea de date în momentul apariției frontului crescător de ceas.

12.2.1. Registre cu încărcare asincronă:

– Comanda de memorare se dă printr-un front crescător pe linia comună a intrărilor de ceas; semnalul de încărcare LOAD nu este un ceas propriu-zis (periodic și permanent), ci apare doar o singură dată.

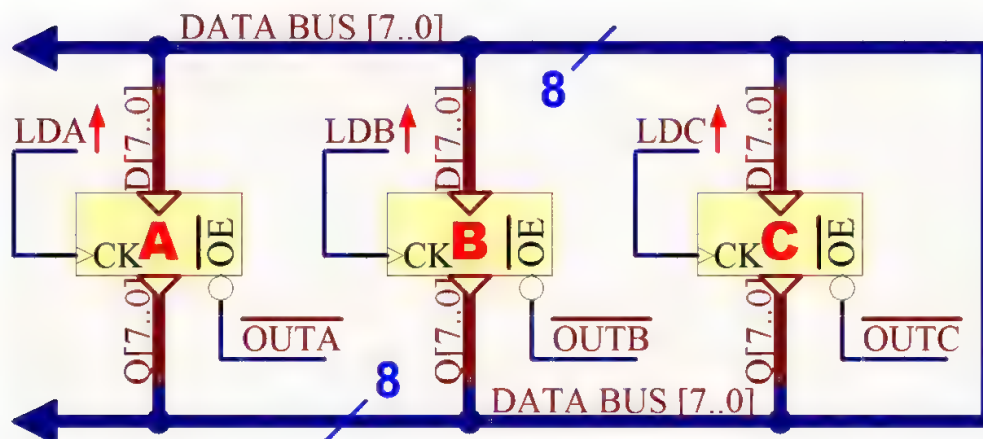


– Ieșirile bistabilelor sunt trecute prin “buffer”-e cu 3 stări validate de semnalul comun

$\boxed{\overline{OE} = \text{"Output Enable"}}$, pentru a putea fi conectate direct la o magistrală.

Observații: – În interiorul procesoarelor din sistemele de calcul se găsesc **blocuri de registre** având intrările și ieșirile conectate împreună la aceeași magistrală comună, pentru a se putea transfera date între ele;

– Dacă registrele sunt cu **încărcare asincronă**, comanda de încărcare vine separat pentru fiecare registru în parte și nu există nici o posibilitate de sincronizare a momentului încărcării printr-un semnal de ceas comun.

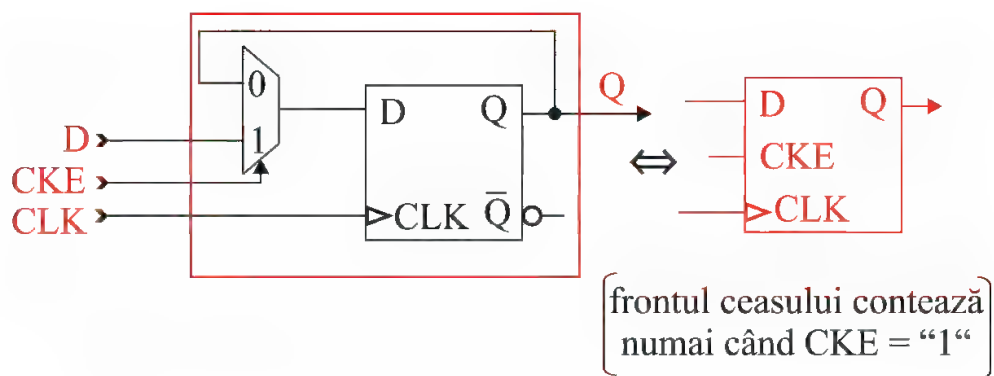


12.2.2. Registre cu încărcare sincronă:

– Comanda de încărcare a datelor în registru se dă printr-o intrare auxiliară de **validare a ceasului** **CKE = "Clock Enable"** și nu printr-un front pe intrarea de ceas propriu-zisă.

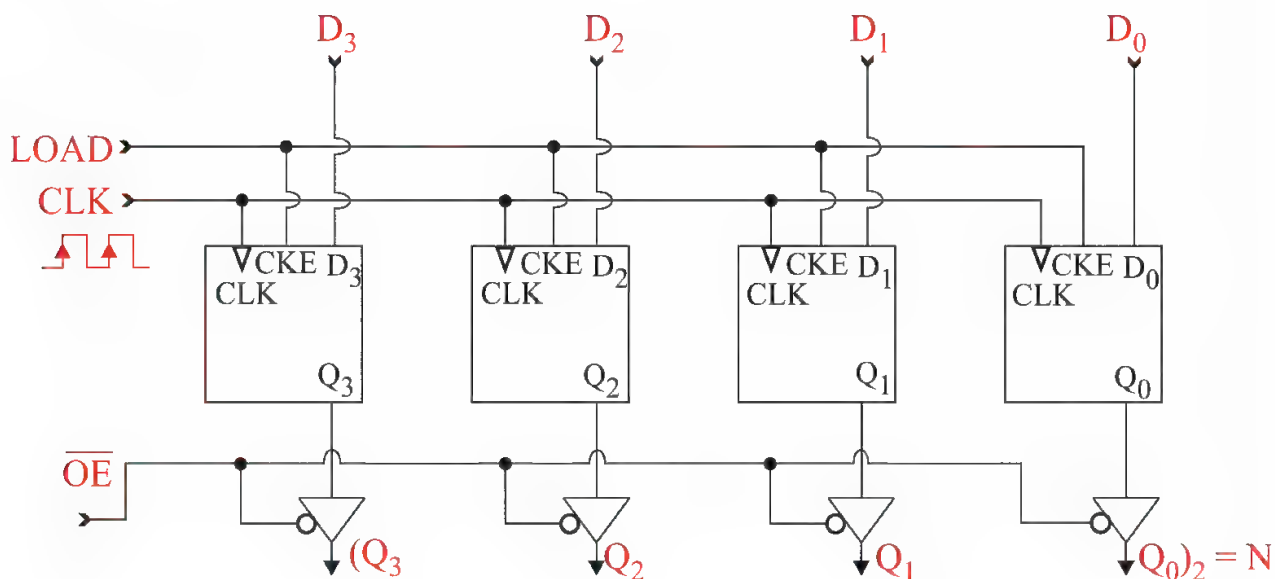
– Încărcarea are loc tot în momentul frontului de ceas, dar numai dacă CKE = "1"; dacă în momentul respectiv semnalul CKE = "0", frontul ceasului este ignorat și registrul își păstrează valoarea anterioară.

– Pentru a obține intrarea auxiliară de validare a ceasului, fiecare bistabil **D** al registrului trebuie modificat prin inserarea pe intrare a unui **multiplexor 2:1** care selectează între intrarea de date externă sau propria ieșire, în funcție de semnalul CKE:



CKE = $\begin{cases} 0 & \Rightarrow \text{la fiecare front crescător se re-memorează ieșirea ;} \\ 1 & \Rightarrow \text{la fiecare front crescător încarcă intrarea ;} \end{cases}$

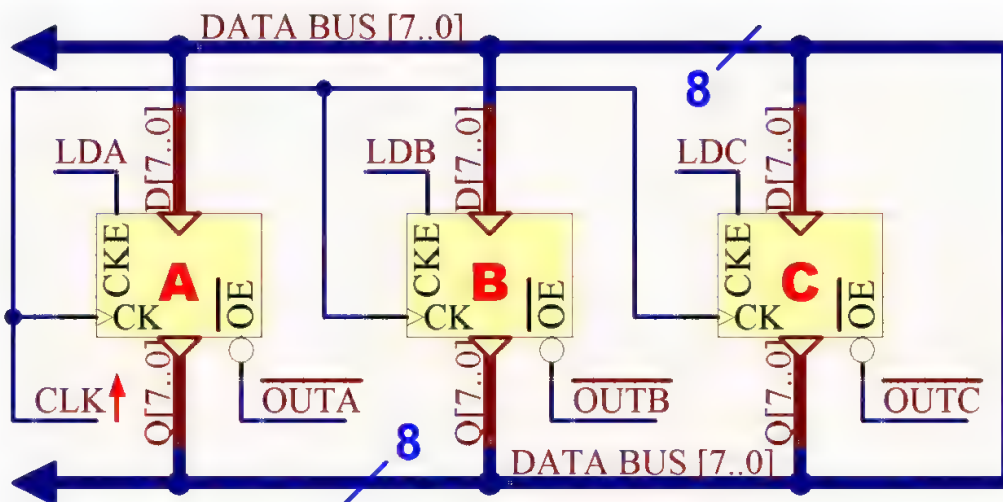
– Efectul obținut este cel de ignorare a frontului de ceas, chiar dacă în realitate bistabilul tot se încarcă în momentul frontului, dar cu aceeași valoare pe care o avea și înainte.



– Biții numărului aplicat la intrare se memorează în registru prin comanda **LOAD = CKE = "1"**, dar numai în momentul următorului front pozitiv al ceasului CLK.

Observații: – Astfel de registre se numesc cu **încărcare sincronă**, deoarece momentul încărcării poate fi precis sincronizat prin frontul semnalului de ceas comun tuturor registrelor din cadrul blocului;

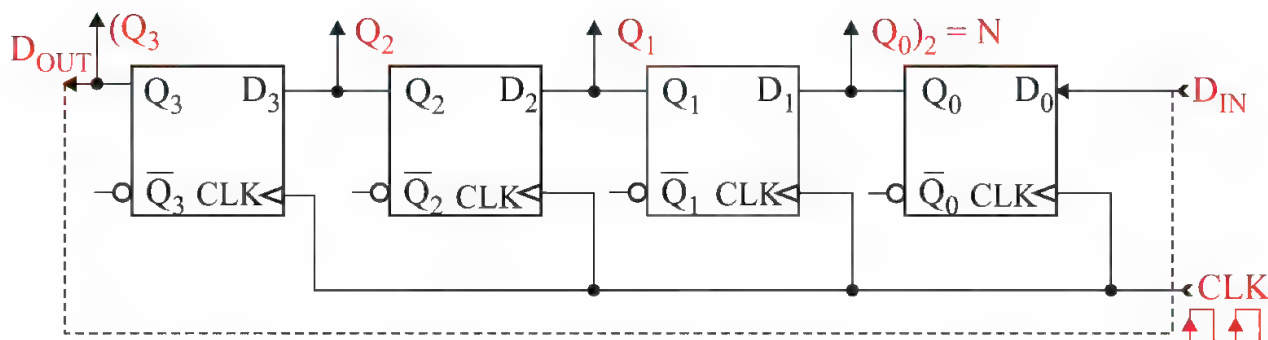
– Chiar dacă **ceasul este comun, periodic și permanent**, comanda de încărcare va selecta numai un singur registru din ansamblu și numai un singur front, prin menținerea validării CKE pe “1” logic numai la acel registru și numai pe durata unei singure perioade de ceas.



12.2.3. Registre de deplasare / rotație:

– Permit deplasarea biților între rangurile binare ale numărului, cu sau fără recirculare a transportului (rotație);

– Intrarea unui bistabil **D** se conectează la ieșirea celui anterior:



se conectează în buclă pentru rotație

– Fiecare bistabil **D** se încarcă cu valoarea care era conținută în cel anterior, în mod simultan, sincron cu frontul crescător al ceasului comun $\Rightarrow$ în fiecare perioadă a ceasului se realizează deplasarea spre stânga cu o poziție binară a numărului memorat în registru.

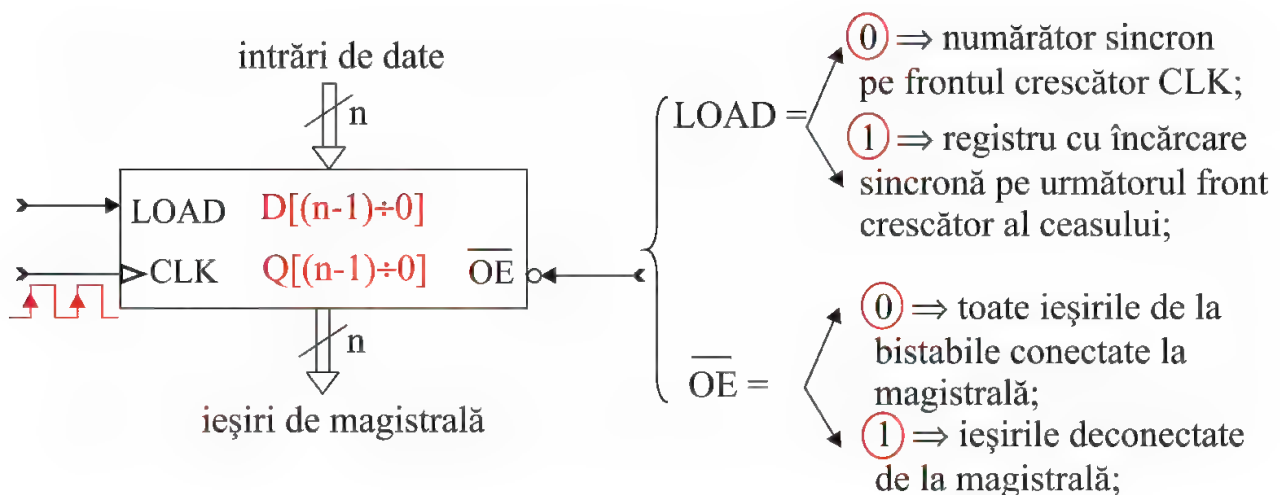
– Pentru deplasarea spre dreapta, se realizează conexiunile în ordine inversă:

$$(D_0 = Q_1, D_1 = Q_2, D_2 = Q_3, D_3 = D_{IN}, D_{OUT} = Q_0).$$

12.3. Numărătoare cu încărcare

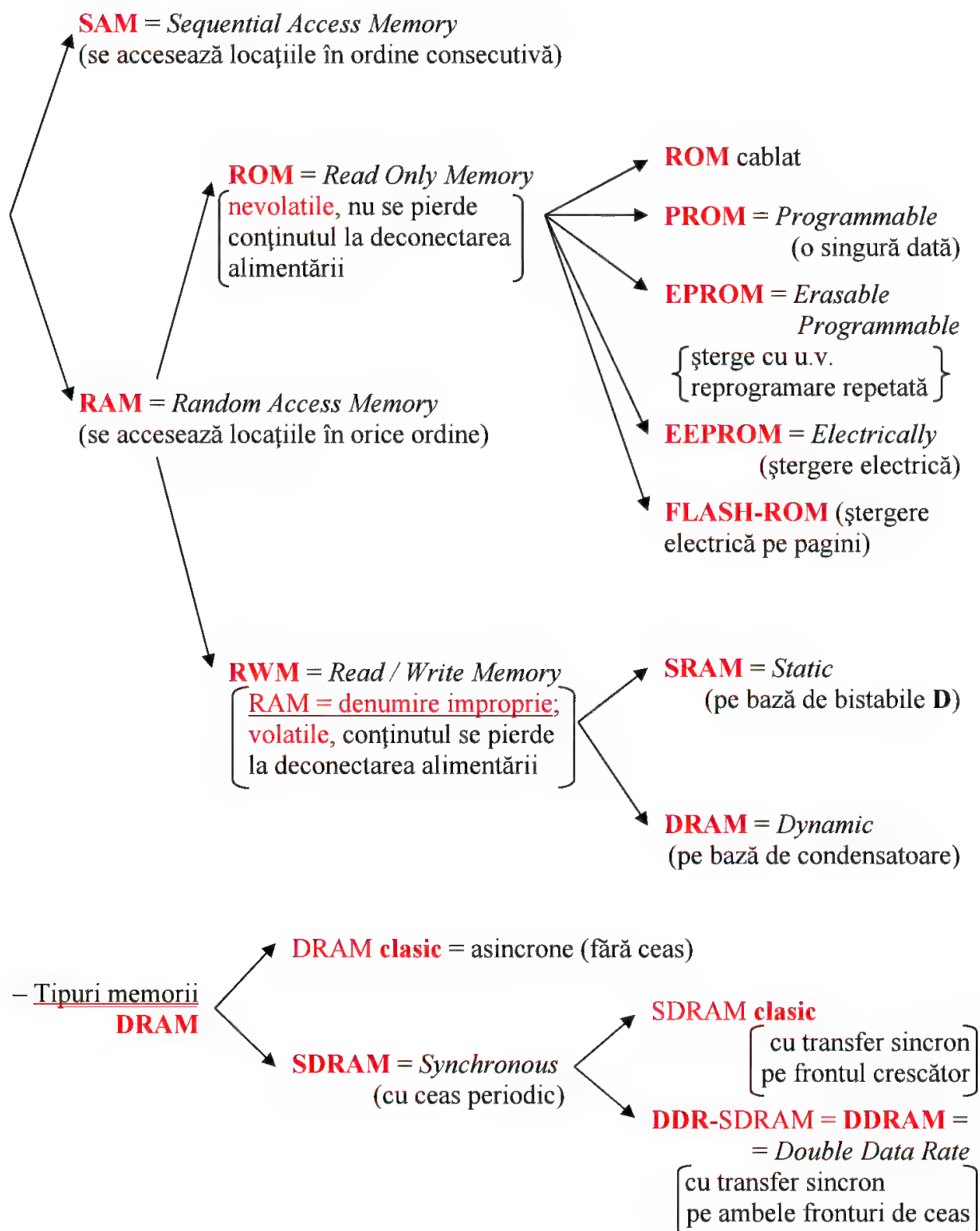
– Sunt structuri secvențiale care combină funcționarea de tip numărător cu funcționarea ca registru de memorare.

– Se construiesc prin inserarea unor multiplexoare înainte de intrările fiecărui bistabil, care să comute modul lor de conectare dinspre exterior sau de la ieșirile celorlalte ranguri; comanda de selecție LOAD este comună tuturor multiplexoarelor și conduce la formarea unei structuri de tip registru când are valoarea “1” logic, sau a unei structuri de tip numărător sincron când are valoarea “0”.



13. Memoria sistemelor de calcul

– Tipuri de memorii în funcție de tehnologie:

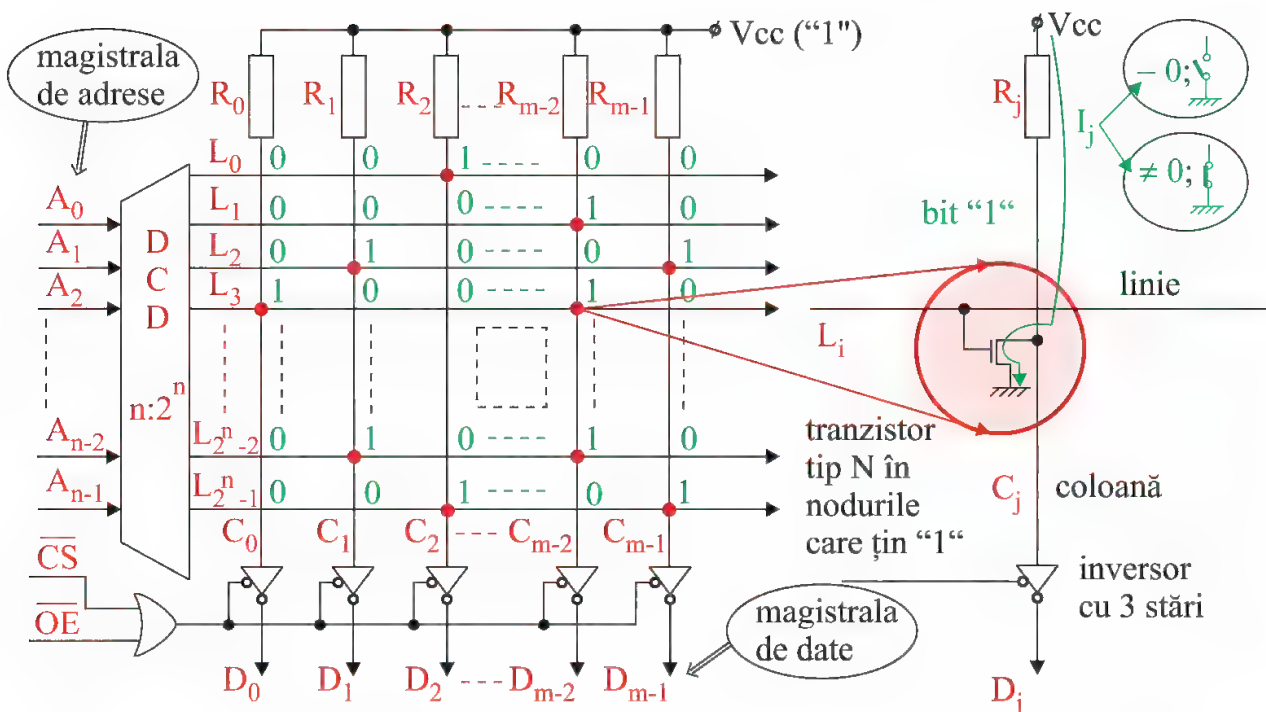


Observație: – Memoriile RAM dinamice (**DRAM**) sunt construite pe bază de condensatoare, care se descarcă singure în timp și trebuie re-împrospătate periodic, chiar dacă sunt alimentate permanent cu tensiune.

13.1. Memorii de tip ROM

– Sunt structuri **combi-naționale** organizate sub formă de matrice cu

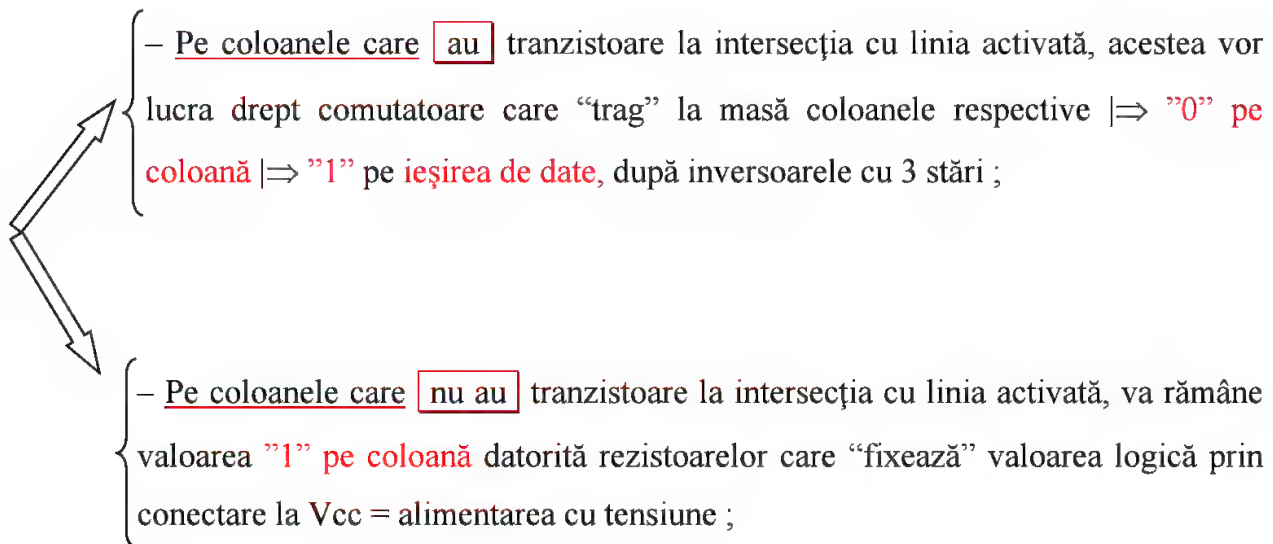
2^n linii x **m** coloane, în fiecare nod al matricei fiind memorat câte un bit.



– **Construcție:** – În nodurile matricei care memorează biți de "1" se găsesc tranzistoare tip N care "trag" coloanele respective la "0" logic când pe linia respectivă se primește "1" logic.

– În nodurile care memorează biți de "0", aceste tranzistoare lipsesc prin construcție, ori au fost deconectate prin programarea sau arderarea unor siguranțe.

– **Funcționare:** – Decodorul **D**CD **$n:2^n$** va pune pe "1" logic o singură linie din matrice la un moment dat, în funcție de combinația binară a biților magistralei de adresă $(A_{n-1}, \dots, A_2, A_1, A_0)_2$, care definește adresa locației selectate = linia activată pe "1";

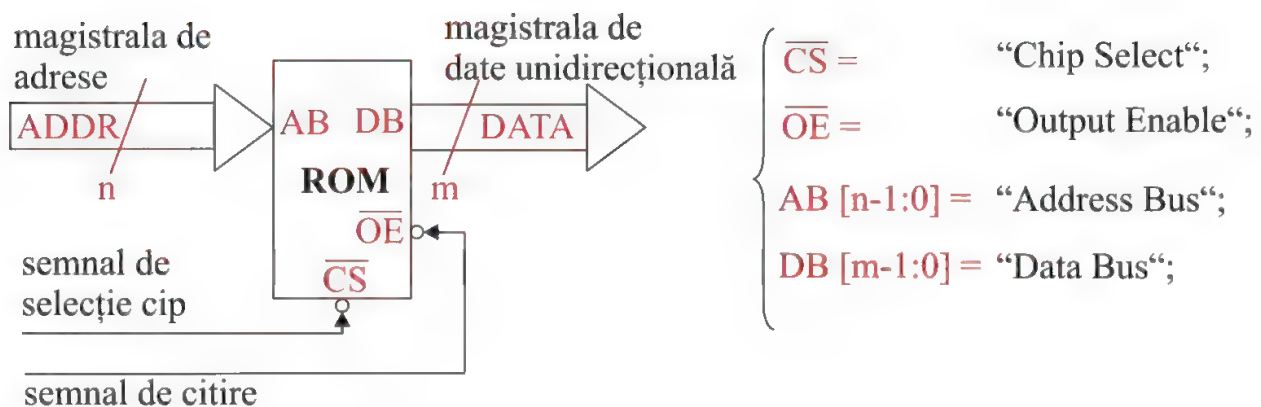


– Toate celelalte linii ale decodurului în afară de cea activată la “1” vor avea “0” logic în acel moment $\Rightarrow$ locațiile sunt **neselectate** și **nu vor avea nici un efect asupra coloanelor**, indiferent dacă au sau nu tranzistoare, pentru că acestea nu se vor mai închide la masă.

Observație: – Rezistoarele R_j de pe fiecare coloană au rolul de a “fixa” la “1” logic coloana în lipsa conectării la masă a tranzistoarelor, dar și rolul de a limita curentul I_j prin tranzistoare de la V_{cc} la masă când acestea sunt închise, altfel apare scurtcircuit:

$$I_j = \frac{V_{cc}}{R_j}, \text{ conform legii lui Ohm } .$$

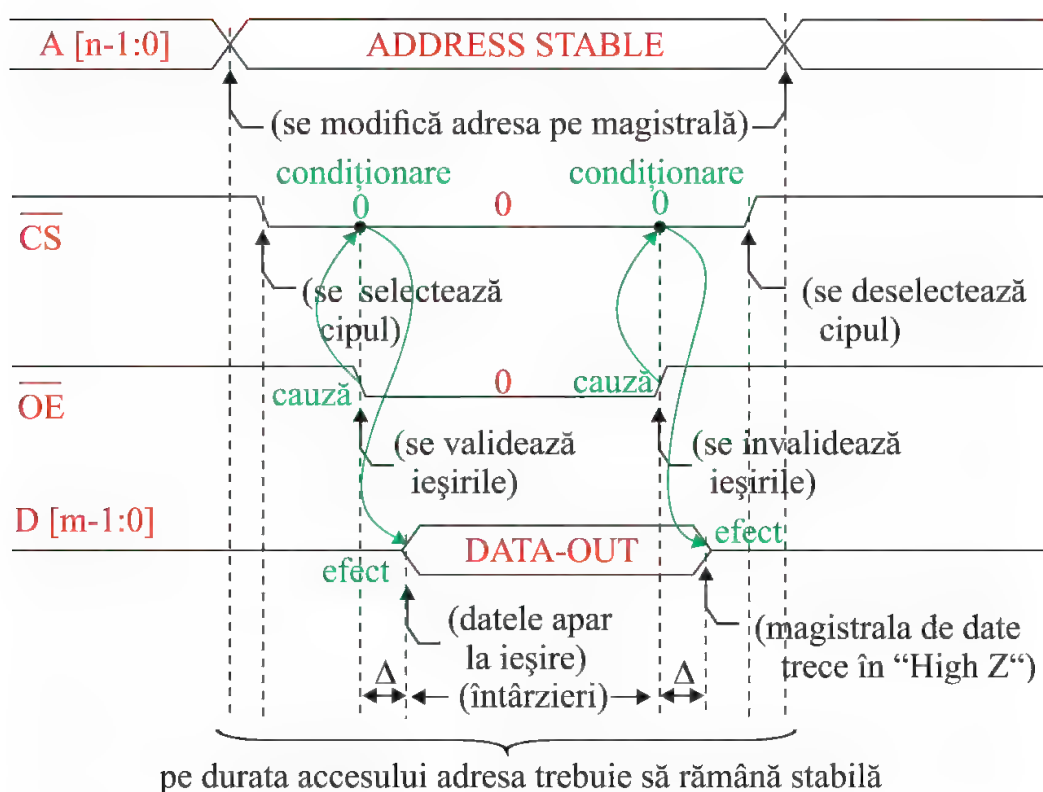
– **Simbol echivalent:**



– Tabel de adevăr:

$\overline{CS}$	$\overline{OE}$	DB [m-1:0]	Obs.
1	–	Ieșiri deconectate HZ	Cip neselectat
0	1	Ieșiri deconectate HZ	Ieșire nevalidată
0	0	Citește DOUT	READ

Observație: – Biții din locația selectată prin combinația de adrese trec la ieșiri numai când: $\overline{CS} = 0$ și $\overline{OE} = 0$, altfel ieșirile sunt deconectate.

– Diagramă de timp:– Capacitatea memoriei:

Fiecare locație (linie) memorează câte un cuvânt de **m** biți care se citesc simultan la ieșiri ;
 În total sunt **2^n** locații, fiecare memorând un cuvânt

⇒

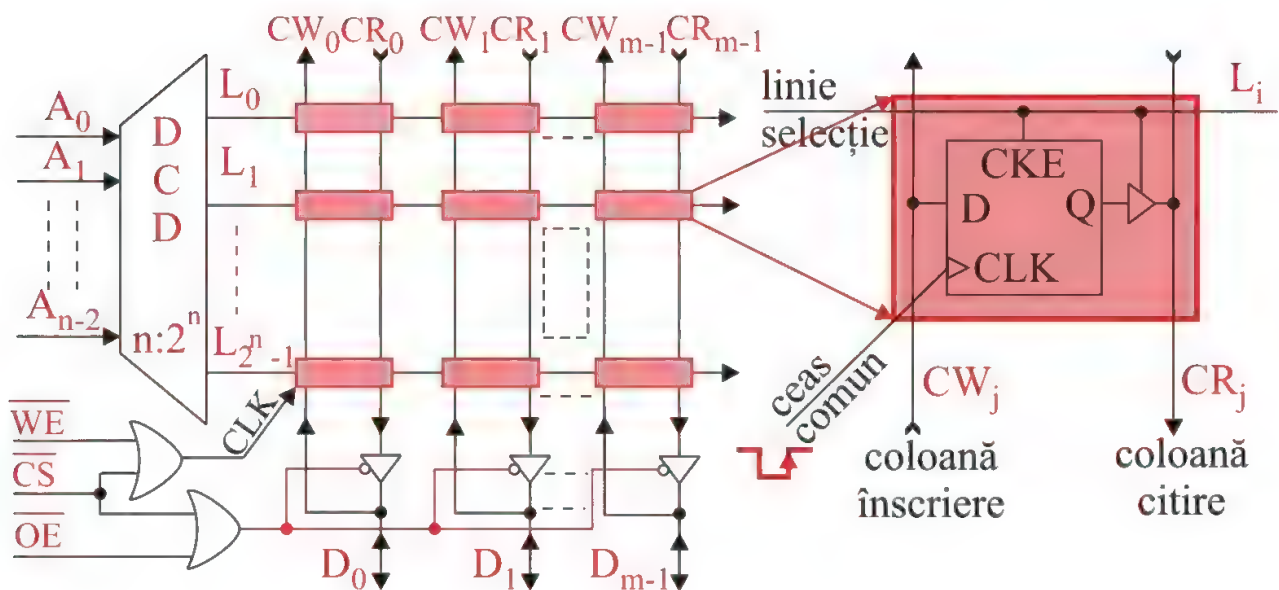
⇒ Rezultă capacitatea de stocare: **$C = 2^n \cdot m$ biți**.

– **Exemplu:** $\left. \begin{matrix} n = 11 \\ m = 8 \end{matrix} \right\} \Rightarrow C = 2^{11} \cdot 8 \text{ biți} = 2 \cdot 2^{10} \cdot 8 \text{ biți} = 2 \cdot 2^{10} \text{ octeți} =$

$$\text{multipli} \left\{ \begin{array}{l} 8 \text{ biți} = 1 \text{ octet} \\ 2^{10} = 1 \text{ Kilo} \\ 2^{10} K = 2^{20} = 1 \text{ Mega} \\ 2^{10} M = 2^{30} = 1 \text{ Giga} \\ 2^{10} G = 2^{40} = 1 \text{ Terra} \end{array} \right\} \Rightarrow = 2 \text{ K octeți}.$$

13.2. Memorii de tip **SRAM**

– Memoriile citește-scrie statice sunt structuri **secvențiale** organizate sub formă de matrice cu 2^n linii x 2^m coloane, la intersecția fiecărei linii cu câte 2 coloane fiind plasat un bistabil de tip **D** cu validare a ceasului și a ieșirii, care memorează valoarea unui bit:



– **Construcție:** – Fiecare linie din matrice formează un registru de date cu m biți, cu validare a ceasului pentru înscrisere (**CKE**) și validare a ieșirii pentru conectarea la coloana comună de ieșire (**OE**, activ pe “1”);

– Coloanele de ieșire sunt conectate prin intermediul unor “buffer”-e cu 3 stări la coloanele de intrare, formând liniile de citire / scriere ale magistralei de date;

– Semnalul de ceas este comun tuturor bistabilelor din matrice și este controlat prin intrarea $\overline{WE}$ = "Write Enable" la memoriile SRAM asincrone.

– **Funcționare:** – Decodorul de adresă **DCD n:2ⁿ** activează pe "1" o singură linie din matrice, corespunzătoare locației selectate prin combinația respectivă a biților de adresă ($A_{n-1}, A_{n-2}, \dots, A_1, A_0$)₂; toate celelalte linii sunt inactive, în "0";

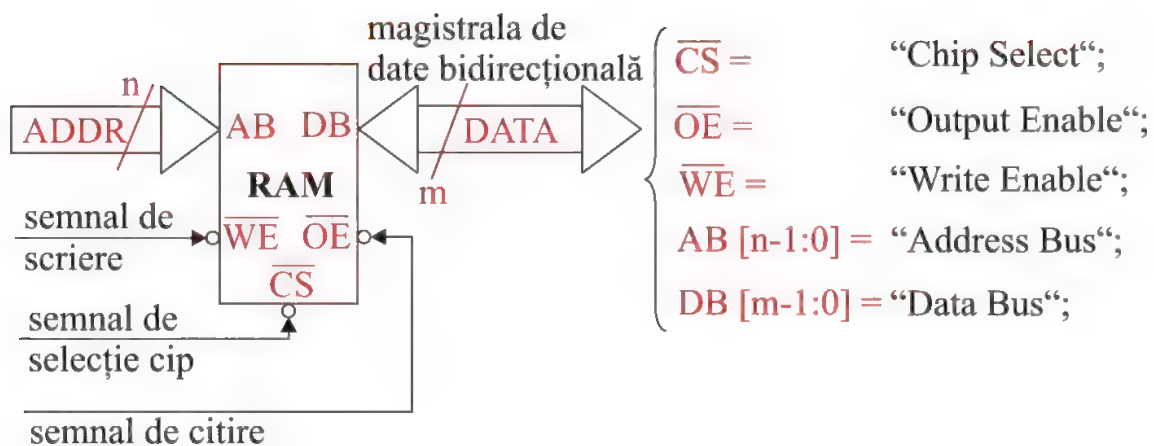
– Linia activă validează atât intrarea de control a scrierii **CKE** = "Clock Enable" a fiecărui bistabil de pe linie, cât și "buffer"-ul de conectare a bistabilului la coloana de citire;

– Semnalul de selecție al cipului $\overline{CS}$ condiționează atât citirea $\overline{OE} = 0$ cât și scrierea $\overline{WE} = 0$ în toate bistabilele de pe linia (locația) selectată; $\overline{CS} = 0$ pentru ca scrierea și citirea să poată avea loc;

– La activarea în "0" logic a semnalului de scriere, datele au fost deja depuse pe magistrala de date și sunt păstrate stabile pe toată durata cât $\overline{WE} = 0$; frontul pozitiv apărut la dezactivarea lui $\overline{WE}$ produce înscrierea datelor în toate bistabilele de pe aceeași linie selectată;

– La activarea în "0" logic a semnalului de scriere, "buffer"-ele cu 3 stări de la ieșirile coloanelor de citire sunt validate și toate bistabilele de pe linia selectată își transmit valorile de la ieșiri pe magistrala de date.

– **Simbol echivalent:**



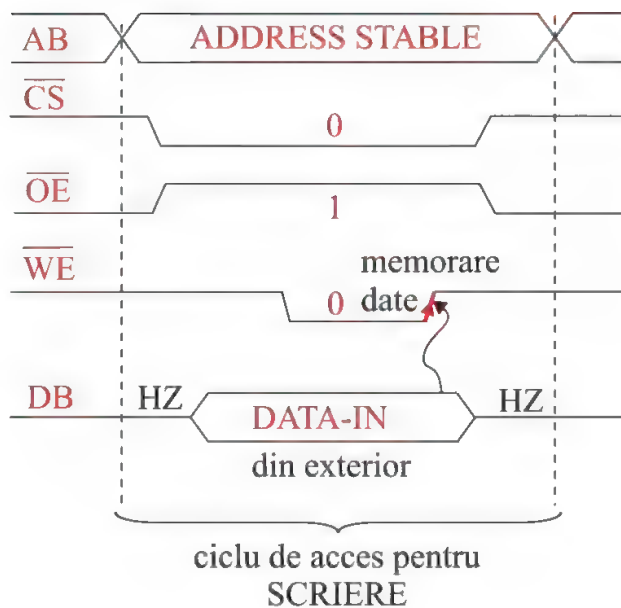
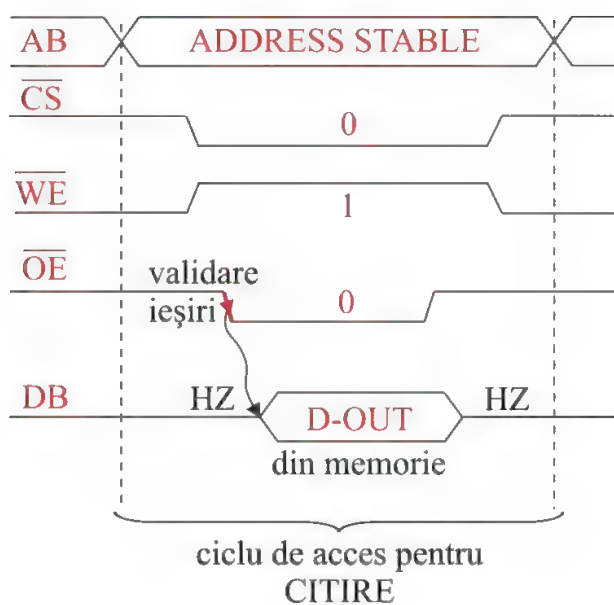
– Tabel de adevăr:

$\overline{CS}$	$\overline{WE}$	$\overline{OE}$	DB [m-1:0]		Obs.
			IN	OUT	
1	–	–	Nu se înscrie	HZ	INACTIV
0	1	1	Nu se înscrie	HZ	INACTIV
0	1	0	Nu se înscrie	Citește DOUT	READ
0	0	1	Înscrie DIN	HZ	WRITE
0	0	0	DIN	DOUT	INVALID (se înscrie citirea)

Observații:

– Dacă ambele semnale $\overline{WE}$ și $\overline{OE}$ sunt activate în “0” și cipul este selectat, se reînscrie informația citită de pe linie;

– Unele memorii au $\overline{OE}$ prioritar față de $\overline{WE}$ și permit doar citirea !

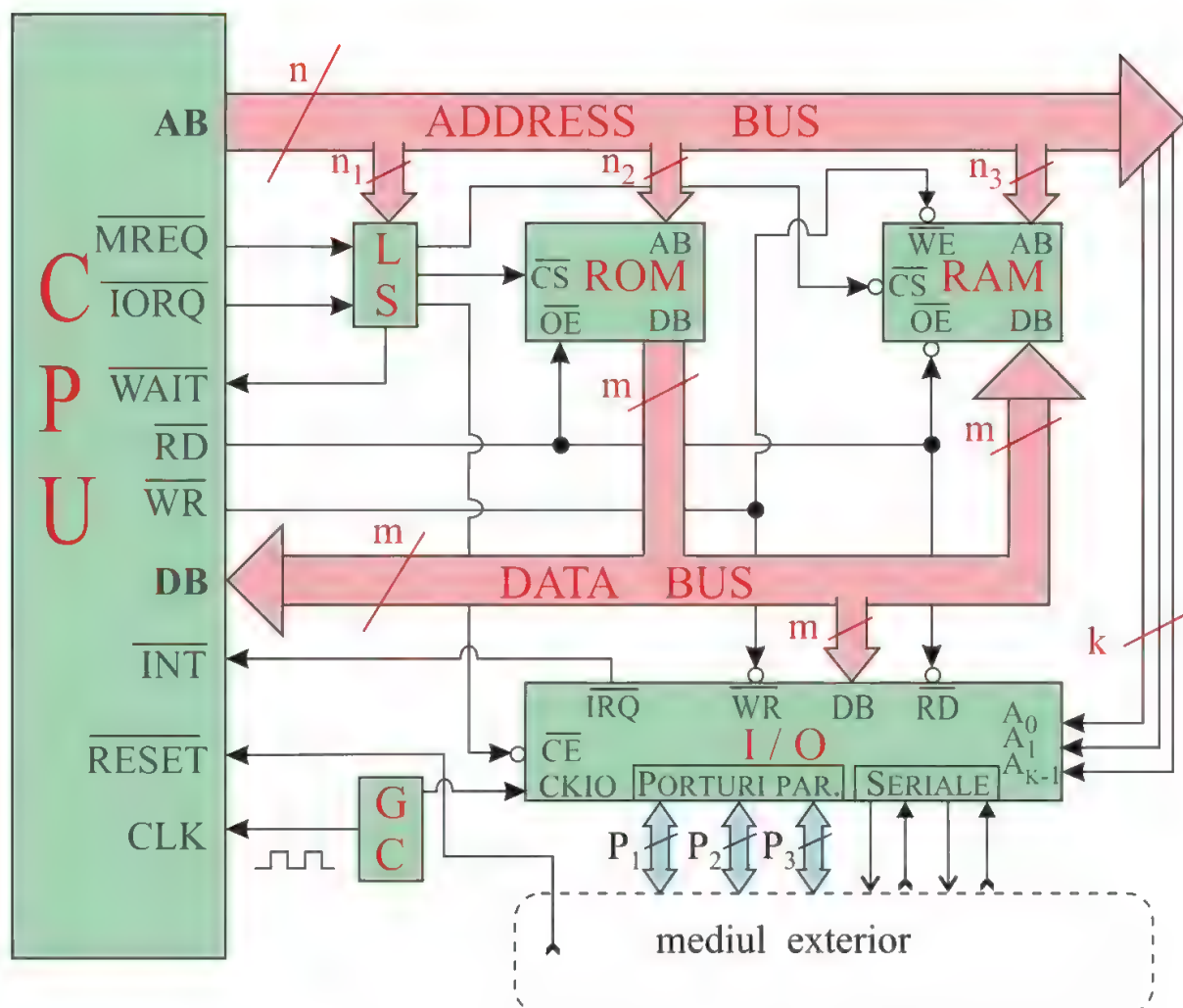
– Diagrame de timp:

14. Organizarea unui sistem de calcul

– Un sistem de calcul trebuie să conțină întotdeauna 3 categorii de componente principale:

- unitatea centrală de procesare (microprocesorul);
 $\mu P = CPU = \text{"Central Processing Unit"};$
- memoriile volatile și nevolatile (RAM și ROM);
- structurile logice de intrare / ieșire:
 $I/O = \text{"Input / Output"}.$

– Schema - bloc arată modul general de interconectare al acestora:

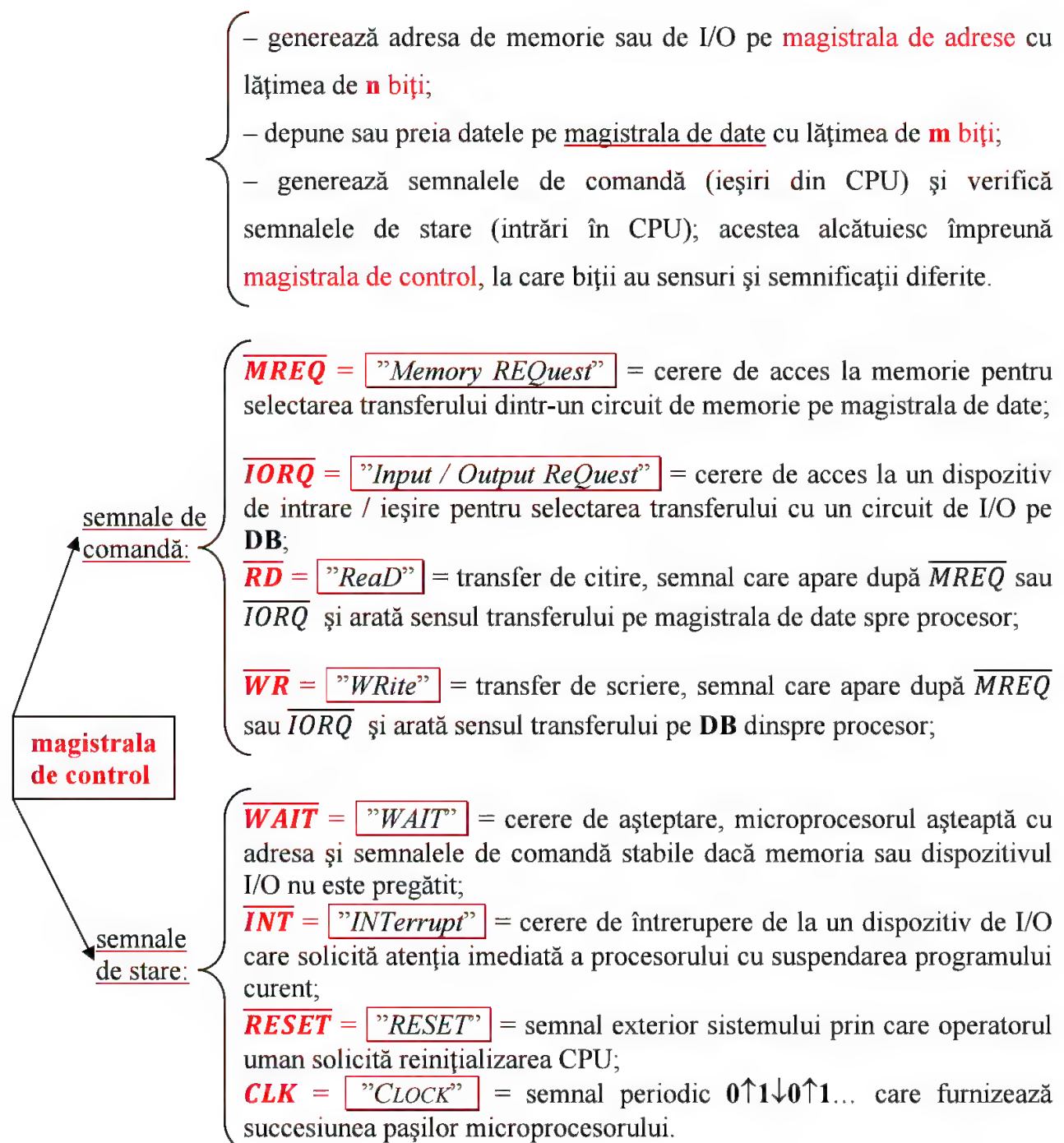


– Pe lângă cele 3 structuri principale, sistemul de calcul trebuie să mai conțină și alte circuite, care formează măcar încă 2 blocuri funcționale:

$$\begin{cases} \text{LS} = \text{logica de selecție a circuitelor de memorie și I/O;} \\ \text{GC} = \text{generatorul de ceas periodic pentru microprocesor.} \end{cases}$$

– Singurul circuit care generează comenzi în sistem este microprocesorul; celelalte dispozitive doar ascultă de comenzi și răspund.

– **Microprocesorul:**



Observații: – Semnalele $\overline{MREQ}$ și $\overline{IORQ}$ care specifică tipul accesului nu apar niciodată simultan cu valoarea “0”, ci sunt exclusive;

– Uneori aceste semnale sunt reunite împreună sub forma unei linii:

$$IO/\overline{M} = \begin{cases} "0" \Rightarrow \text{acces memorie;} \\ "1" \Rightarrow \text{acces intrare / ieșire;} \end{cases}$$

– Semnalele $\overline{RD}$ și $\overline{WR}$ care specifică sensul transferului nu apar niciodată active simultan pe “0”, ci sunt exclusive;

– Uneori aceste semnale sunt reunite împreună sub forma unei linii:

$$W/\overline{R} = \begin{cases} "0" \Rightarrow \text{citire date;} \\ "1" \Rightarrow \text{scriere date.} \end{cases}$$

– Operații pe magistrala de date:

$\overline{IORQ}$	$\overline{MREQ}$	$\overline{WR}$	$\overline{RD}$	OPERAȚIE
1	1	1	1	Internă CPU
1	0	1	0	Citire din memorie
1	0	0	1	Scriere în memorie
0	1	1	0	Citire din I/O
0	1	0	1	Scriere în I/O

– Întotdeauna după trecerea în “0” a lui $\overline{IORQ}$ sau $\overline{MREQ}$ urmează trecerea în “0” a lui $\overline{WR}$ sau $\overline{RD}$, așa că nu există și alte combinații în tabelul de adevăr.

– Logica de selecție:

– primește semnalele de comandă de la CPU și activează (pe zero) unul singur dintre semnalele de

selecție a cipurilor $\left\{ \begin{matrix} \overline{CS} = \text{"Chip Select"} \\ \overline{CE} = \text{"Chip Enable"} \end{matrix} \right\}$;

– selecția are loc și ținând cont de o parte $n_1 \ll n$ din magistrala de adrese;

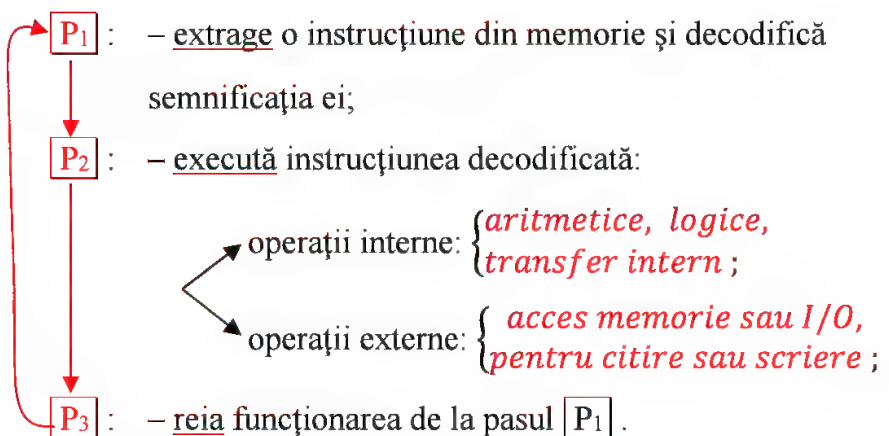
– activează semnalul de așteptare $\overline{WAIT}$ către procesor dacă viteza de acces e prea mare;

- **Memoria:**
- primește o parte $n_2 < n$ (ROM) sau $n_3 < n$ (RAM) din biții magistralei de adrese și transferă datele folosind toți cei m biți ai magistralei de date;
 - primește semnalele de citire $\overline{RD} \rightarrow \overline{OE}$ (care validează “buffer”-ele de ieșire) și de scriere $\overline{WR} \rightarrow \overline{WE}$ (care înscrie datele în locația selectată);
 - un singur circuit de memorie primește semnalul de selecție $\overline{CS} = 0$ la un acces, în funcție de **restul** n_1 al biților din magistrala de adrese care intră în logica de selecție și nu intră în circuitele de memorie;

– **Dispozitivele de intrare ieșire:**

- permit comunicarea cu mediul exterior prin intermediul **porturilor** paralele și seriale; porturilor de numărare;
- un port paralel de ieșire este format dintr-un registru de date de m biți care memorează valoarea depusă de procesor pe magistrala de date;
- un port paralel de intrare este format dintr-un set de m “buffer”-e cu 3 stări care permit trecerea semnalelor din exterior către magistrala de date pentru a fi citite de procesor;
- un port serial de intrare / ieșire este format dintr-un registru de deplasare de m biți ai cărui bistabili pot fi și citați și înscriși separat pe magistrala de date;
- un port de numărare este format dintr-un numărător care poate contoriza evenimente **externe** sau perioadele ceasului de la generator **CKIO** și generează întreruperi pe linia **IRQ** către CPU; numărătorul este cu încărcare și poate fi scris sau citit de procesor pe magistrala de date;
- o parte mică din biții de adresă $k < n$ este folosită împreună cu semnalul $\overline{CE} = \text{“Chip Enable”}$ pentru selectarea **unui singur port** cu care procesorul să comunice la un acces I/O.

– **Funcționarea microprocesorului:**

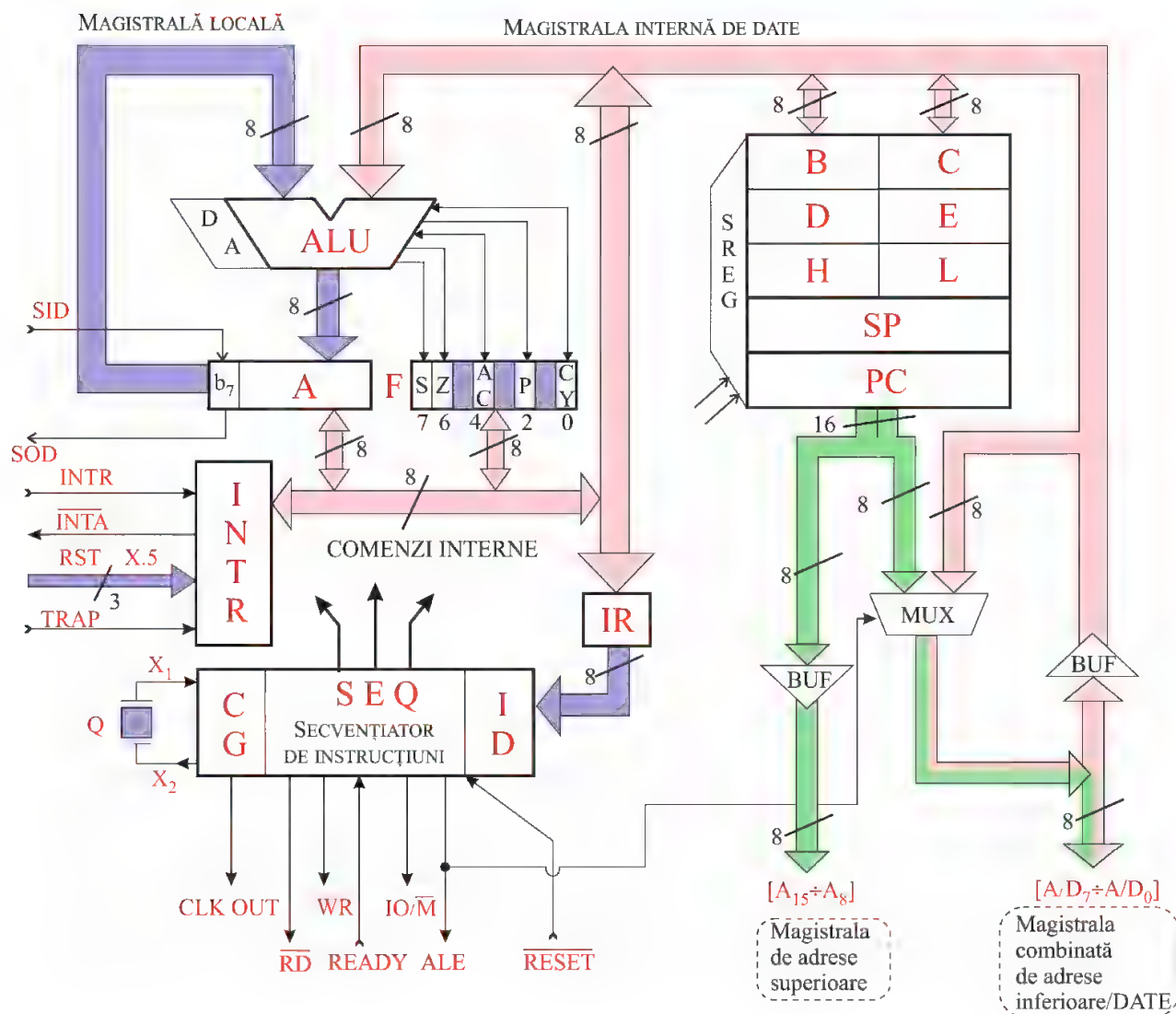


15. Microprocesorul 8085

– Este o unitate centrală de procesare pe **8 biți**, fiind organizat în jurul unei unități aritmetico-logice și al unei magistrale interne de date, ambele pe 8 biți.

15.1. Arhitectura internă

– Schema-bloc a elementelor funcționale într-o variantă simplificată este următoarea:

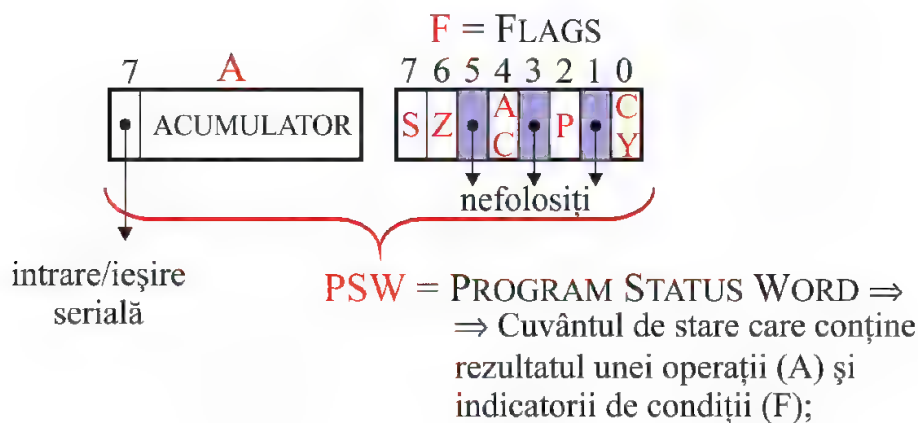


– Unitatea Aritmetico-Logică ALU:

- Este organizată în jurul unui sumator de 8 biți, circuite de multiplexare și porți;
- Efectuează operații *aritmetice* (adunare, scădere, comparație) și *logice* (NOT, AND, OR, XOR), operații de *rotație* cu și prin *CARRY*.
- Unul din operanzi trebuie să se afle în **REGISTRUL ACUMULATOR - A**, unde este depus și rezultatul !

ALU = ARITHMETIC AND LOGIC UNIT ;

DA = DECIMAL ADJUST $\Rightarrow$ Circuite de corecție automată a rezultatului după operații cu numere în cod BCD;



S = SIGN

$\Rightarrow$ Arată semnul rezultatului (copie bitul 7 din A)

dacă s-a lucrat cu numere cu semn ;

Z = ZERO

$\Rightarrow$ Arată că în urma unei operații s-a obținut un rezultat zero în acumulator;

AC = AUXILIARY CARRY

$\Rightarrow$ Indicator de transport între biții **3 și 4** din acumulator, folosit numai de blocul de ajustare zecimală (**DA**);

P = PARITY

$\Rightarrow$ Arată că în urma unei operații s-a obținut în acumulator un număr **par** de biți "1";

Cy = CARRY

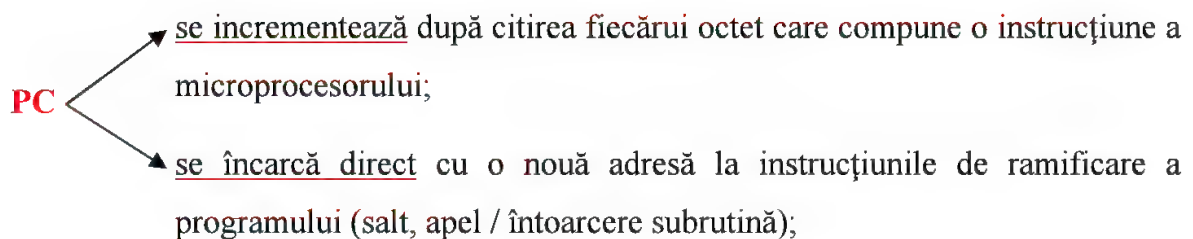
$\Rightarrow$ Indică transportul din sau spre bitul 7 al acumulatorului și este utilizat la rotații.

Observație: – Bitul 7 din acumulator poate fi încărcat din - sau copiat în - exteriorul microprocesorului pe liniile de transfer serial:

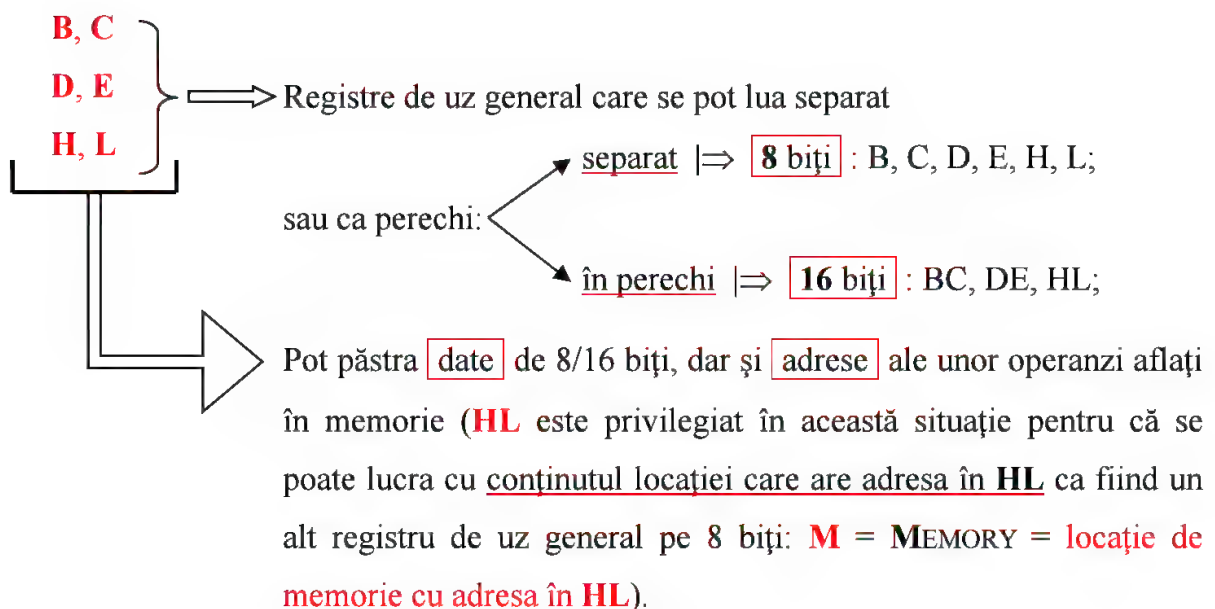
SID = SERIAL INPUT DATA și **SOD** = SERIAL OUTPUT DATA.

– **Blocul de registre:** – Conține registre de uz general și speciale care sunt selectate cu ajutorul unui multiplexor **SREG** și pot fi încărcate de pe magistrala internă de date, iar conținutul lor poate fi transmis atât pe magistrala de date internă cât și spre magistrala de adrese.

PC = PROGRAM COUNTER $\Rightarrow$ **REGISTRU SPECIAL ESENȚIAL**, care păstrează adresa următoarei instrucțiuni din memorie; este un numărător cu posibilitate de încărcare:



SP = STACK POINTER $\Rightarrow$ Registru special care păstrează adresa unei zone de memorie denumită **STIVĂ**, unde se țin date și adrese, cu acces printr-un singur capăt;



– **Blocul de secvențiere și control:** **SEQ** = SEQUENCER $\Rightarrow$ are rolul de a coordona succesiunea de comenzi interne și externe necesare pentru execuția instrucțiunilor;

IR = INSTRUCTION REGISTER $\Rightarrow$ Registru al secvențiatorului care păstrează codul pe 8 biți al instrucțiunii care a fost citită din memorie și urmează a fi executată;

ID = INSTRUCTION DECODER $\Rightarrow$ Decodificator special care are rolul de a selecta succesiunea de operații necesare la execuția unei instrucțiuni, în funcție de codul din IR;

CG = CLOCK GENERATOR $\Rightarrow$ Acest microprocesor are generatorul de ceas încorporat și necesită doar un **crystal de quartz** conectat în exterior de la ieșirea **X₂** spre intrarea **X₁**, cu rol în stabilizarea perioadei ceasului; generatorul furnizează și ceasul pentru dispozitivele de intrare / ieșire, pe linia externă **CLKOUT**;

– Blocul de secvențiere și control furnizează semnalele externe pentru comanda sistemului de calcul; o parte din ele sunt:

$\overline{RD}$

= **READ**, semnal de citire pe magistrala externă de date;

$\overline{WR}$

= **WRITE**, semnal de scriere pe magistrala externă de date;

$IO/\overline{M}$

= **IO / MEMORY ACCESS**, semnal obținut prin unificarea semnalelor exclusive $\overline{MREQ}$ și $\overline{IORQ}$:

$IO/\overline{M} \begin{cases} = 0 \Rightarrow \overline{MREQ} = 0, \text{ acces pentru memorie;} \\ = 1 \Rightarrow \overline{IORQ} = 0, \text{ acces pentru intrare / ieșire;} \end{cases}$

(unificarea a fost posibilă pentru că $\overline{MREQ}$ și $\overline{IORQ}$ nu apar niciodată simultan)

READY

= $\overline{WAIT}$ = semnal echivalent cu $\overline{WAIT}$ negat, care face procesorul să aștepte la acces dacă memoria sau dispozitivul I/O selectat nu este pregătit pentru transfer;

$READY \begin{cases} = 0 \Rightarrow \overline{WAIT} = 0, (activ), \text{ dispozitivul nu este pregătit, CPU așteaptă;} \\ = 1 \Rightarrow \overline{WAIT} = 1, (inactiv), \text{ dispozitivul este "gata" pentru transfer, CPU nu așteaptă;} \end{cases}$

$\overline{RESET}$

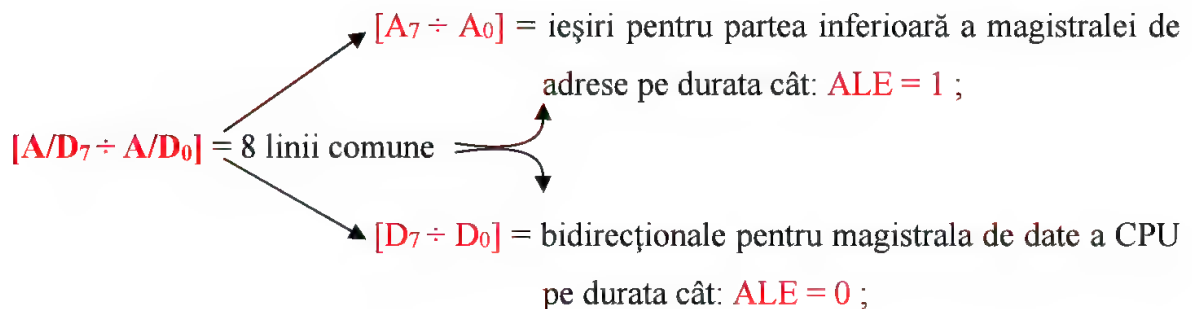
= semnal de reinițializare a procesorului, care aduce în starea inițială secvențiatorul de instrucțiuni și resetează bistabilii din registrul **PC** $\Rightarrow PC = 0000|_{16} = 0000H \Rightarrow$ programul în curs se abandonează și se începe execuția de la instrucțiunea din prima locație a memoriei;

ALE

= **ADDRESS LATCH ENABLE** = semnal care arată că jumătatea inferioară a magistralei de adrese este prezentă pe liniile comune de date și adrese $[A/D_7 \div A/D_0]$, putând fi memorată într-un *registru* sau *latch* extern.

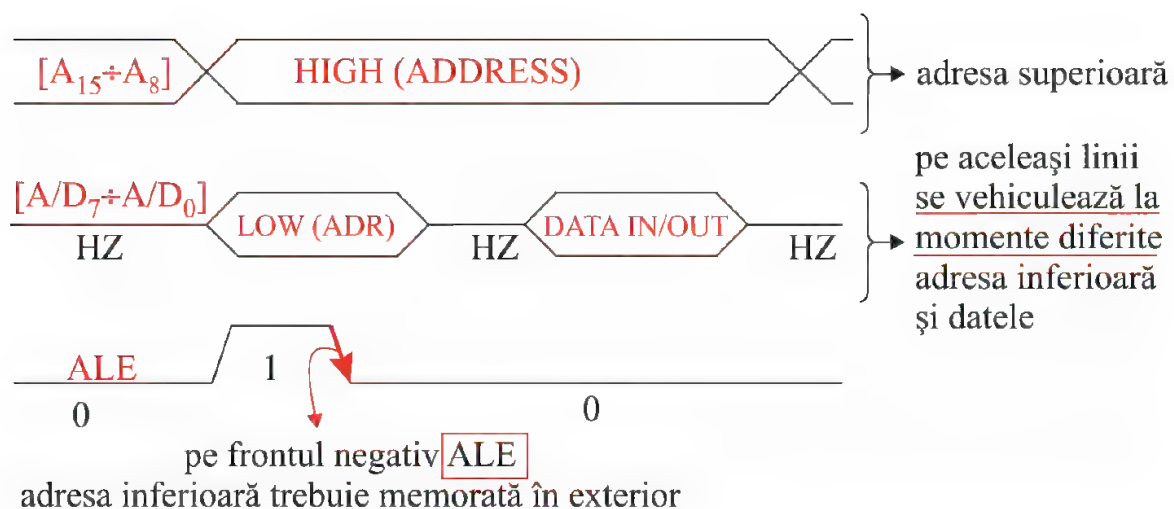
– **Logica magistrelor externe:** – Este formată dintr-o serie de circuite care asigură interfațarea cu magistralele externe:

$[A_{15} \div A_8]$ = partea superioară a magistralei de adrese, formată din 8 linii de ieșire trecute printr-un “buffer” cu HZ;

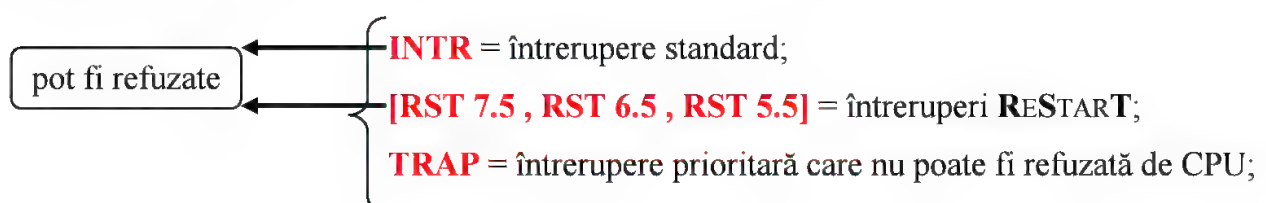


Observație: – Pentru intrarea datelor este folosit un “buffer” de intrare, iar ieșirea datelor este comutată cu ieșirea adreselor, folosind un multiplexor comandat de ALE.

– **Diagrama de timp a unui acces extern:**



– **Logica de întreruperi INTR:** – O întrerupere este un eveniment extern care solicită atenția procesorului prin activarea unui semnal de intrare.



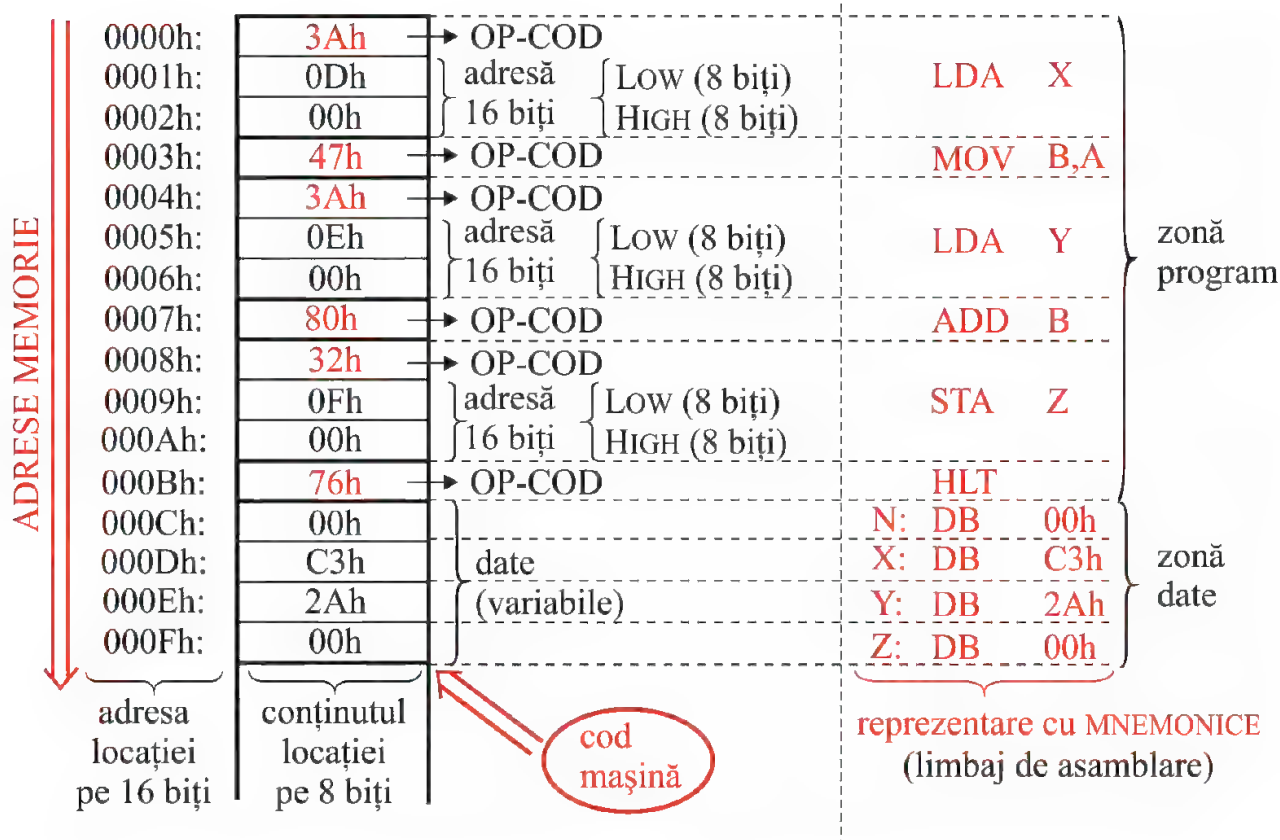
– Procesorul semnalează acceptarea întreruperii printr-un semnal de ieșire: **INTA** = **INTERRUPT ACKNOWLEDGE** și începe execuția unui sub-program special de tratare a evenimentului.

15.2. Funcționarea microprocesorului

– Instrucțiunile microprocesorului **8085** pot fi de **1**, **2** sau **3** octeți:

- primul octet este prezent întotdeauna și reprezintă codul operațiilor de executat: **OP-CODE** ;
- următorii **1** sau **2** octeți de după OP-CODE sunt opționali și reprezintă **CONSTANTE** sau **ADRESE** pe 8 sau pe 16 biți .

– Considerăm un program care realizează calculul sumei a două numere de 8 biți fără semn, aflate în 2 locații de memorie, apoi depune rezultatul într-o a 3-a locație:



– Execuția programului pe pași: – Inițial: $PC = 0000h$;

P 1 : $IR = M(PC) = M(0000h) = 3Ah$; în registrul de instrucțiuni se încarcă conținutul locației de memorie de la adresa PC;

P 2 : $DECODE =$ Decodificarea $\Rightarrow$ codul $3Ah$ cere să se încarce registrul acumulator cu conținutul adresei de memorie care urmează după op-cod ; trebuie extrasă și adresa;

P 3 : $PC = PC + 1 = 0001h$; se incrementează PC-ul;

P 4 : $LOW = M(PC) = M(0001h) = 0Dh$; în registrul temporar LOW se încarcă cei 8 biți mai puțin semnificativi ai adresei;

P 5 : $PC = PC + 1 = 0002h$; se incrementează PC-ul;

P 6 : $HIGH = M(PC) = M(0002h) = 00h$; în registrul temporar HIGH se încarcă cei 8 biți mai semnificativi ai adresei operandului;

P 7 : $PC = PC + 1 = 0003h$; se incrementează PC-ul;

P 8 : $A = M(HIGH, LOW) = M(00h, 0Dh) = M(000Dh) = C3h$; se depune pe magistrala de adrese ansamblul registrelor temporare concatenate (HIGH, LOW) și se citește conținutul locației de la această adresă pe magistrala de date, încărcându-l în A;

P 9 : $IR = M(PC) = M(0003h) = 47h$; se citește următorul op-cod de la adresa PC și se încarcă în registrul de instrucțiuni;

P10 : $DECODE =$ Decodificarea $\Rightarrow$ codul $47h$ îi spune să copie valoarea din acumulator în registrul de uz general B;

P11 : $PC = PC + 1 = 0004h$; se incrementează PC-ul;

P12 : $B = A = C3h$; execuție internă: copie valoarea din A în B;

P13 : $IR = M(PC) = M(0004h) = 3Ah$;

P14 : $DECODE \Rightarrow$ codul $3Ah$ încarcă în A valoarea de la adresa care urmează;

P15 : $PC = PC + 1 = 0005h$;

P16 : $LOW = M(PC) = M(0005h) = 0Eh$;

P17 : $PC = PC + 1 = 0006h$;

P18 : $HIGH = M(PC) = M(0006h) = 00h$;

P19 : $PC = PC + 1 = 0007h$;

P20 : $A = M(HIGH, LOW) = M(00h, 0Eh) = M(000Eh) = 2Ah$;

-
- P21** : $IR = M(PC) = M(0007h) = 80h$; extrage op-codul;
- P22** : **DECODE** $\Rightarrow$ codul **80h** cere să se depună pe magistrala internă de date conținutul registrului B care să fie adunat cu registrul A prezent la cealaltă intrare a unității ALU, iar rezultatul să fie încărcat în A;
- P23** : $PC = PC + 1 = 0008h$;
- P24** : $A = A + B = 2Ah + C3h = EDh$; execuția internă;
-
- P25** : $IR = M(PC) = M(0008h) = 32h$; extrage op-codul;
- P26** : **DECODE** $\Rightarrow$ codul **32h** cere să se încarce în memorie la adresa care urmează după op-cod valoarea din acumulator;
- P27** : $PC = PC + 1 = 0009h$;
- P28** : $LOW = M(PC) = M(0009h) = 0Fh$;
- P29** : $PC = PC + 1 = 000Ah$;
- P30** : $HIGH = M(PC) = M(000Ah) = 00h$;
- P31** : $PC = PC + 1 = 000Bh$;
- P32** : $M(HIGH, LOW) = A \Rightarrow M(000Fh) = A = EDh$;
-
- P33** : $IR = M(PC) = M(000Bh) = 76h$; op-codul următor;
- P34** : **DECODE** $\Rightarrow$ codul **76h** cere oprirea microprocesorului;
- P35** : $PC = PC + 1 = 000Ch$; pregătește adresa pentru op-codul următor;
- P36** : **STOP** ; microprocesorul se oprește din funcționare.
-

Observație: – Dacă microprocesorul nu s-ar fi oprit din funcționare, ar continua cu extragerea valorilor din zona de date, care **ar fi interpretate drept instrucțiuni** , cu efecte imprevizibile:

- P37** : $IR = M(PC) = M(000Ch) = 00h$; extrage valoarea variabilei N și o consideră drept op-cod;
- P38** : **DECODE** $\Rightarrow$ codul **00h** este un caz fericit în care nu se execută nici o operație:
NOP = No OPERATION;
- P39** : $PC = PC + 1 = 000Dh$;
-
- P40** : $IR = M(PC) = M(000Dh) = C3h$; interpretează valoarea variabilei X ca fiind un cod de instrucțiune;
-

- P41** : DECODE $\Rightarrow$ codul **C3h** este o instrucțiune de salt necondiționat: **JMP = JUMP** la adresa care urmează după op-cod;
- P42** : $PC = PC + 1 = 000Eh$;
- P43** : $LOW = M(PC) = M(000Eh) = 2Ah$; interpretează valoarea variabilei Y ca fiind partea inferioară a adresei de salt;
- P44** : $PC = PC + 1 = 000Fh$;
- P45** : $HIGH = M(PC) = M(000Fh) = EDh$; interpretează rezultatul ca fiind partea superioară a adresei de salt;
- P46** : $PC = PC + 1 = 0010h$;
- P47** : $PC = (HIGH, LOW) = (EDh, 2Ah) = ED2Ah$; execută saltul în program la adresa extrasă după op-cod **PRIN ÎNCĂRCAREA ADRESEI ÎN PC**, cu abandonarea definitivă a valorii anterioare: **0010h**;
-
- P48** : $IR = M(PC) = M(ED2Ah) = ??$; efectul încărcării în registrul PC este saltul în program, pentru că următoarea instrucțiune se va extrage de la o altă adresă decât cea următoare; în cazul de față programul devine necontrolat pentru că nu se cunoaște ce se află în memorie la adresa **ED2Ah** ...

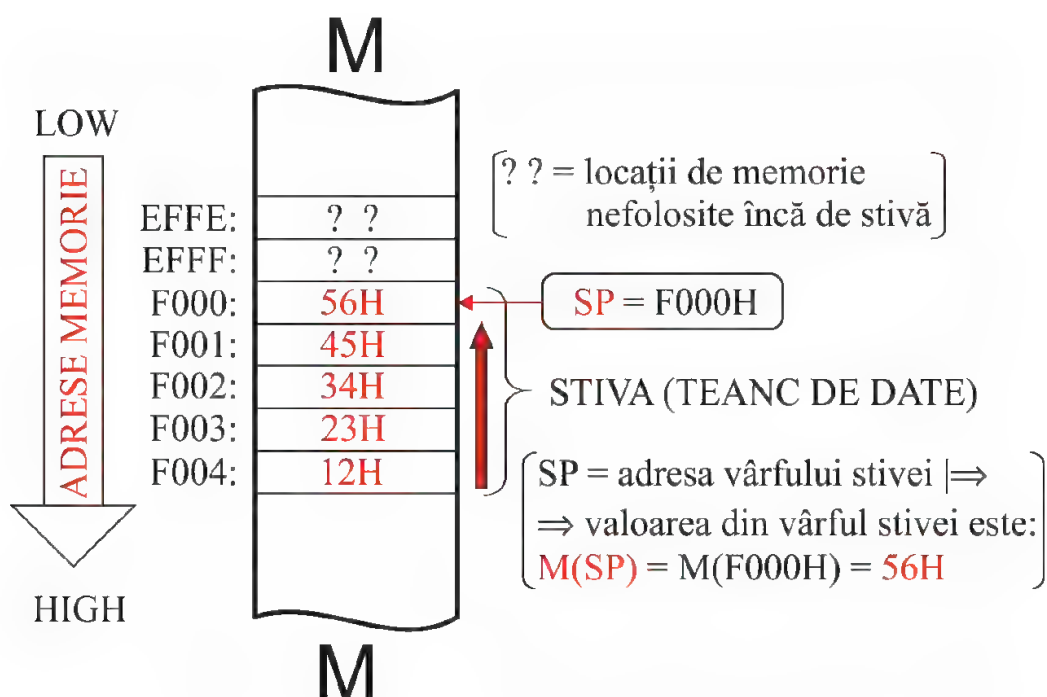
Observatii:

1. Registrele temporare de manevră **HIGH** și **LOW** nu sunt accesibile programatorului, fiind utilizate implicit de către secvențiatorul instrucțiunilor, care dă succesiunea pașilor;
2. Microprocesoarele nu au nici o posibilitate de a face distincția între zonele de memorie ce conțin programul (coduri executabile cu sens) și zonele de memorie ce conțin date (date utile sau valori aleatoare rămase în memorie) ; sarcina rămâne programatorului;
3. Microprocesoarele nu execută de fapt "SALTURI" în program, ci încarcă doar noua valoare a adresei în registrul PC și continuă mecanismul de extragere / incrementare ca și cum nimic nu s-ar fi întâmplat; saltul este *efectul* încărcării în PC;
4. Pentru execuția unei instrucțiuni sunt necesari mai mulți pași mărunți stabiliți de secvențiator prin numărarea perioadelor ceasului;
5. În locul codurilor de instrucțiuni și al constantelor, limbajul de asamblare folosește **MNEMONICE**, care ajută la interpretarea umană, dar trebuie traduse de către ASAMBLOR în coduri înțelese de CPU.

16. Stiva hardware, subrutine și întreruperi

Definiție: – Stiva este o structură de date care ocupă mai multe locații de memorie aflate la adrese consecutive și care este întotdeauna accesată printr-un singur capăt, denumit **VÂRFUL STIVEI**.

– Registrul special indicator de stivă **SP = Stack Pointer** conține întotdeauna adresa ultimei locații ocupate, care constituie vârful stivei:



16.1. Păstrarea temporară a datelor

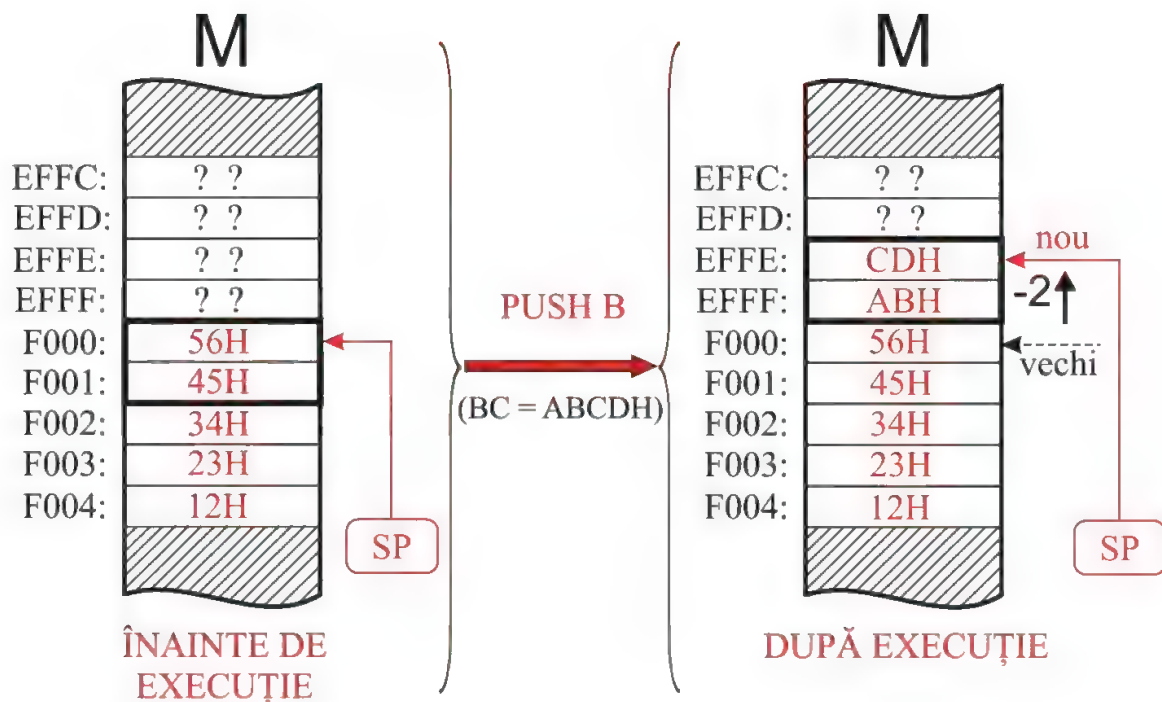
16.1.1. Introducerea datelor în stivă:

– Se face în perechi de câte 2 octeți care sunt încărcăți la adresele imediat inferioare vârfului stivei, prin decrementarea registrului SP.

– Se poate introduce în stivă conținutul registrelor pereche ale microprocesorului, folosind instrucțiuni: **PUSH <reg. pereche>** ;

PUSH PSW (AF) ← perechea acumulator + "flag"-uri;
PUSH B(C) }
PUSH D(E) } ← registrul mai puțin semnificativ
PUSH H(L) } din pereche nu se mai specifică.

Exemplu: BC = ABCDH ; **PUSH B** (faza de execuție a instrucțiunii)

$$\left\{ \begin{array}{l}
 SP = SP - 1 = \text{FFFFH} ; \\
 M(SP) = B \Rightarrow \text{locația } \text{FFFFH} \text{ se încarcă cu valoarea } \text{ABH} ; \\
 SP = SP - 1 = \text{FFFEH} ; \\
 M(SP) = C \Rightarrow \text{locația } \text{FFFEH} \text{ se încarcă cu valoarea } \text{CDH} ;
 \end{array} \right.$$


Observații:

– Introducerea registrelor pereche în stivă are loc în 2 pași consecutivi:

1. Se decrementează SP și se încarcă M (SP) cu registrul mai semnificativ HIGH;
2. Se mai decrementează SP încă o dată și se încarcă registrul LOW la M (SP).

– Decrementarea și înscrierea la adresa SP sunt operații hardware executate automat (fără intervenția programului) $\Rightarrow$ **stivă hardware !**

– Locațiile inferioare vârfului stivei sunt suprascrise cu valorile nou introduse pe stivă.

16.1.2. Extragerea datelor din stivă:

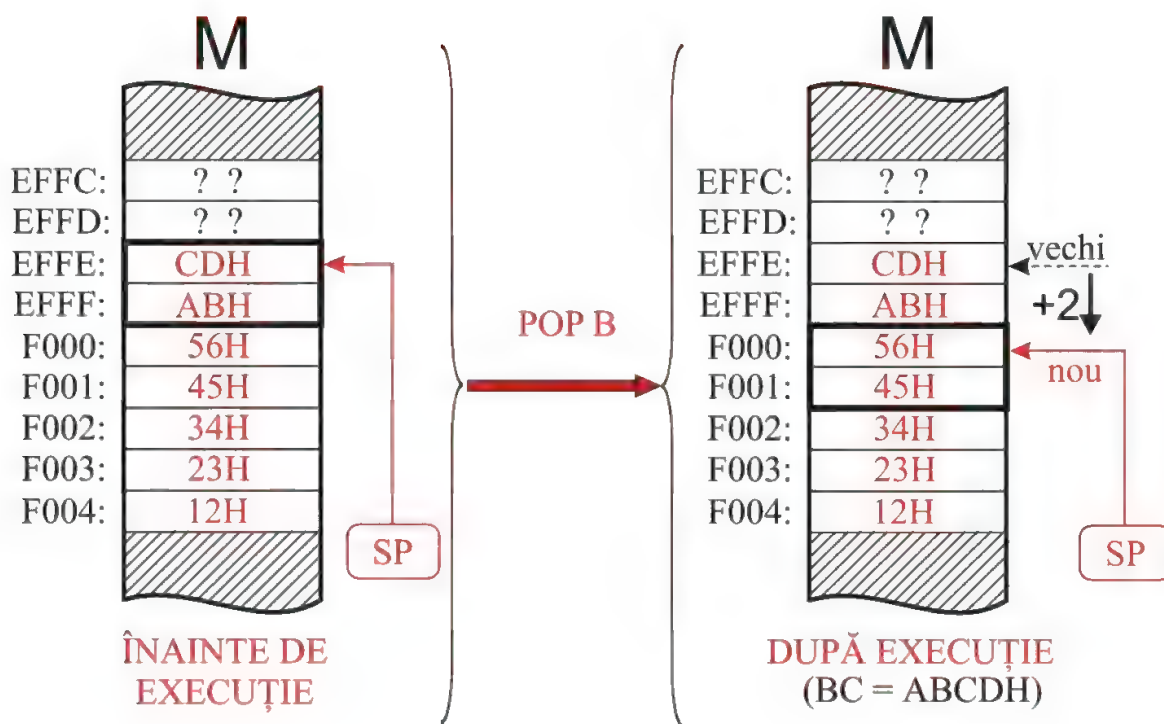
– Se face în perechi de câte 2 octeți care sunt preluați de la adresele din vârful stivei, prin incrementarea consecutivă a registrului SP.

– Se poate extrage din stivă conținutul registrelor pereche ale microprocesorului, folosind instrucțiuni: **POP <reg. pereche>;**

POP H(L) }
POP D(E) } ← registrul mai puțin semnificativ
POP B(C) } din pereche nu se mai specifică;
POP PSW (AF) ← perechea acumulator + "flag"-uri.

– Întotdeauna prima valoare extrasă din stivă va fi ultima valoare introdusă anterior, datorită accesului printr-un singur capăt: {organizare de tip: **LIFO** = LAST-IN, FIRST-OUT}.

Exemplu: **POP B ; BC = ABCDH** (faza de execuție a instrucțiunii)



$$\begin{cases} C = M(SP) \Rightarrow \text{registrul C se încarcă cu valoarea } \boxed{CDH} \text{ de la } \boxed{EFFFH}; \\ SP = SP + 1 = \boxed{EFFFH}; \\ B = M(SP) \Rightarrow \text{registrul B se încarcă cu valoarea } \boxed{ABH} \text{ de la } \boxed{EFFFH}; \\ SP = SP + 1 = \boxed{F000H}; \end{cases}$$

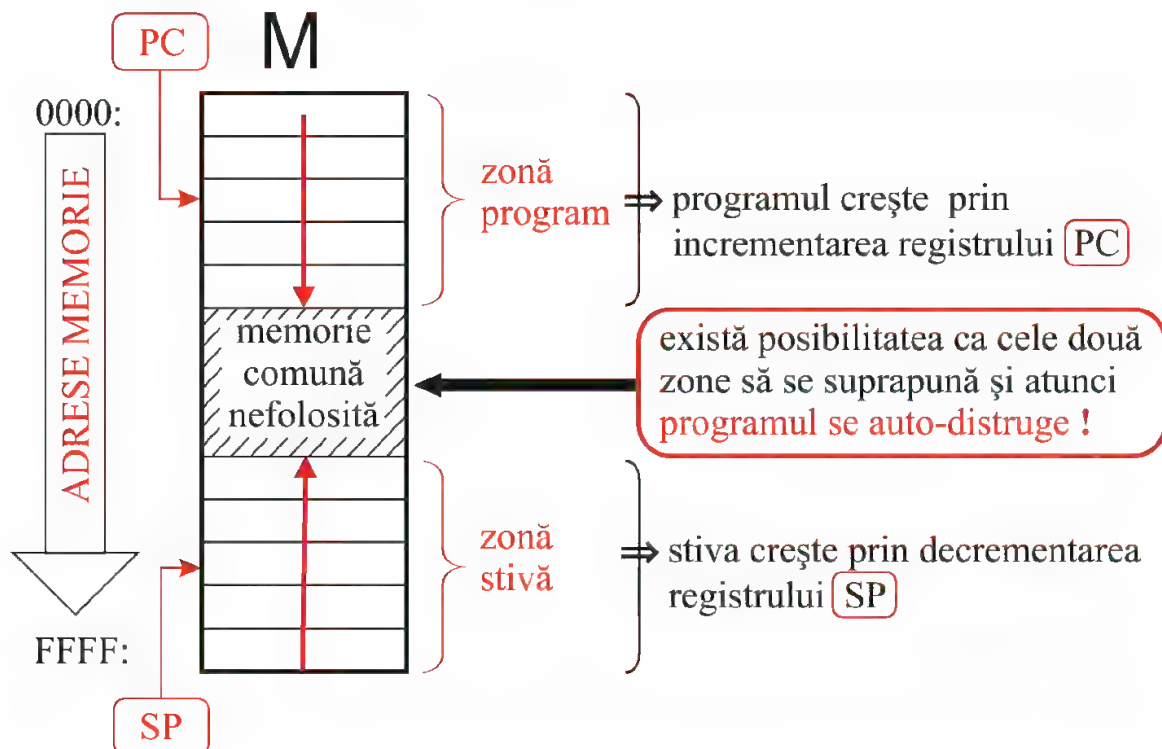
Observații:

- Valorile locațiilor citite de pe stivă rămân **nemodificate**, dar nu mai sunt accesibile datorită mutării indicatorului; ele vor fi suprascrise la următoarea introducere pe stivă.
- Datorită accesului de tip **LIFO**, registrele introduse în stivă se vor extrage obligatoriu în **ordine inversă**.
- Folosind stiva se poate interschimba conținutul a două registre pereche dacă sunt extrase în ordinea introducerii lor:

PUSH B PUSH D POP B POP D	$\Rightarrow BC = \text{vechiul DE}$ $\Rightarrow DE = \text{vechiul BC}$
---	--

- Stiva ocupă locații cu adrese în ordine descrescătoare iar programul ocupă locații cu adrese crescătoare pentru a lăsa între aceste 2 zone o singură parte de memorie neutilizată, care să se poată folosi

$\begin{cases} \text{ori ca stivă;} \\ \text{ori ca program;} \end{cases}$



- Registrul **PC = 0000H** după RESET, dar registrul **SP** trebuie inițializat prin program pentru plasarea stivei la capătul memoriei: **LXI SP,0FFFFH**.
- Stiva se utilizează pentru salvarea comodă a valorii unor registre, fără a fi nevoie să se rețină o adresă fixă de memorie pentru salvarea și refacerea lor.

16.2. Stiva și subrutinele

Definiție: – Se numește **SUBROUTINĂ** (*funcție sau procedură* în limbajele de nivel înalt) o secvență de instrucțiuni ce constituie un subprogram scris o singură dată și executat de cel puțin două ori.

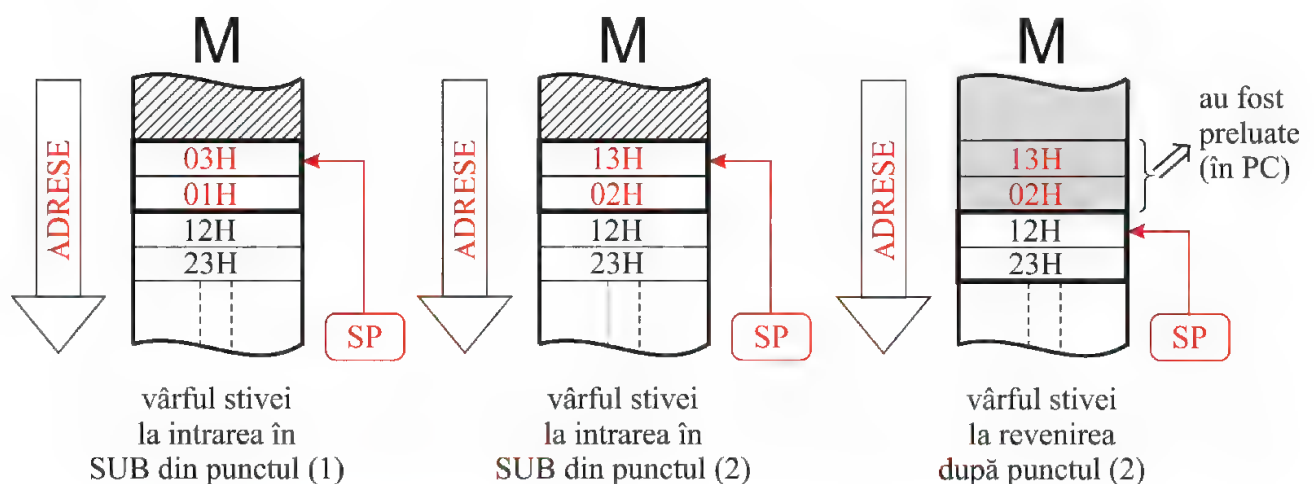
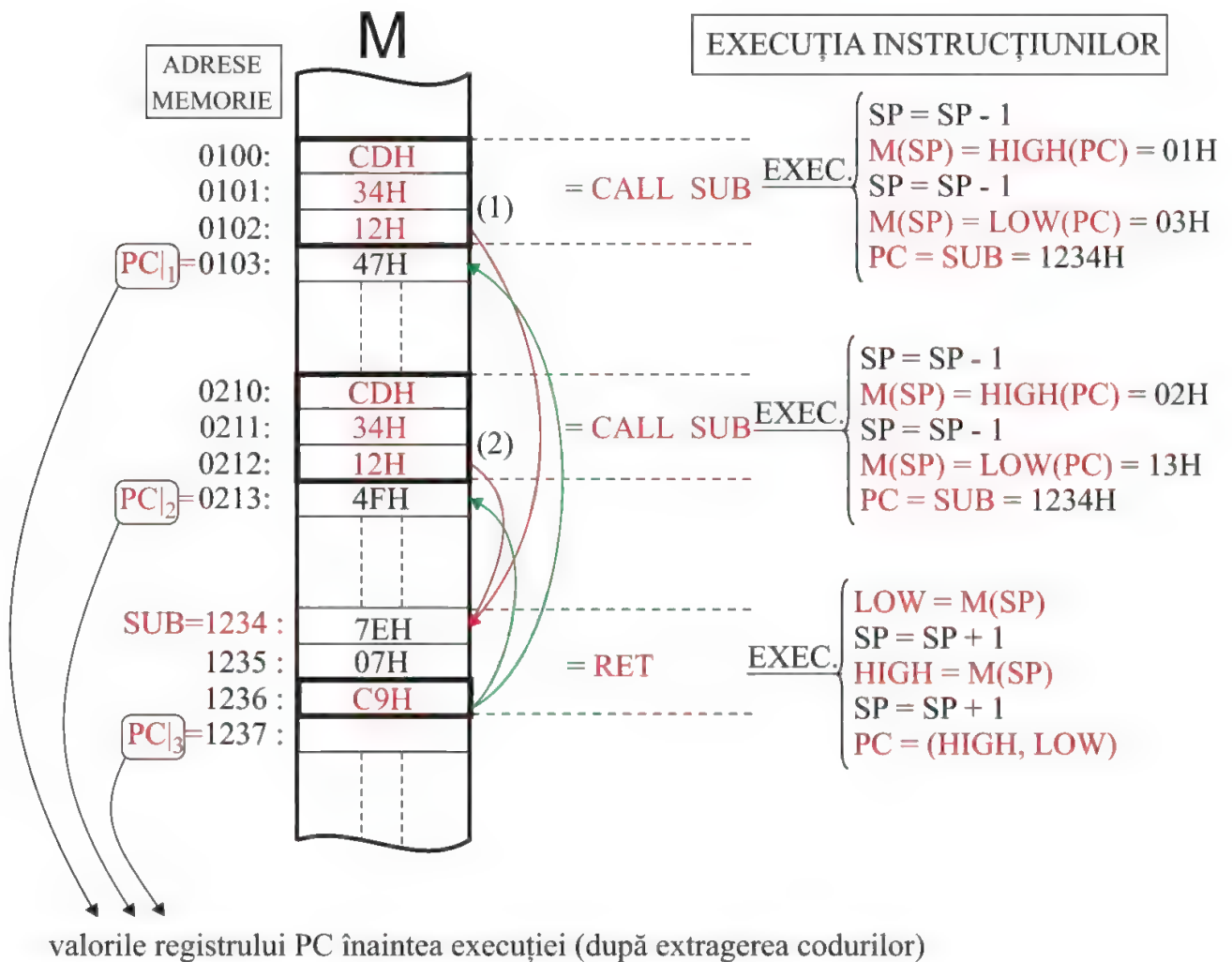
- Intrarea în sub-program (apelul subrutinei) se face cu instrucțiunea:

CALL <etichetă adresă>, iar ieșirea din sub-program (revenirea în programul principal) cu instrucțiunea **RET** = RETURN ; pentru ca fluxul de execuție al programului principal să continue de fiecare dată cu instrucțiunea care urmează apelului CALL, adresa de revenire se salvează automat în stivă.

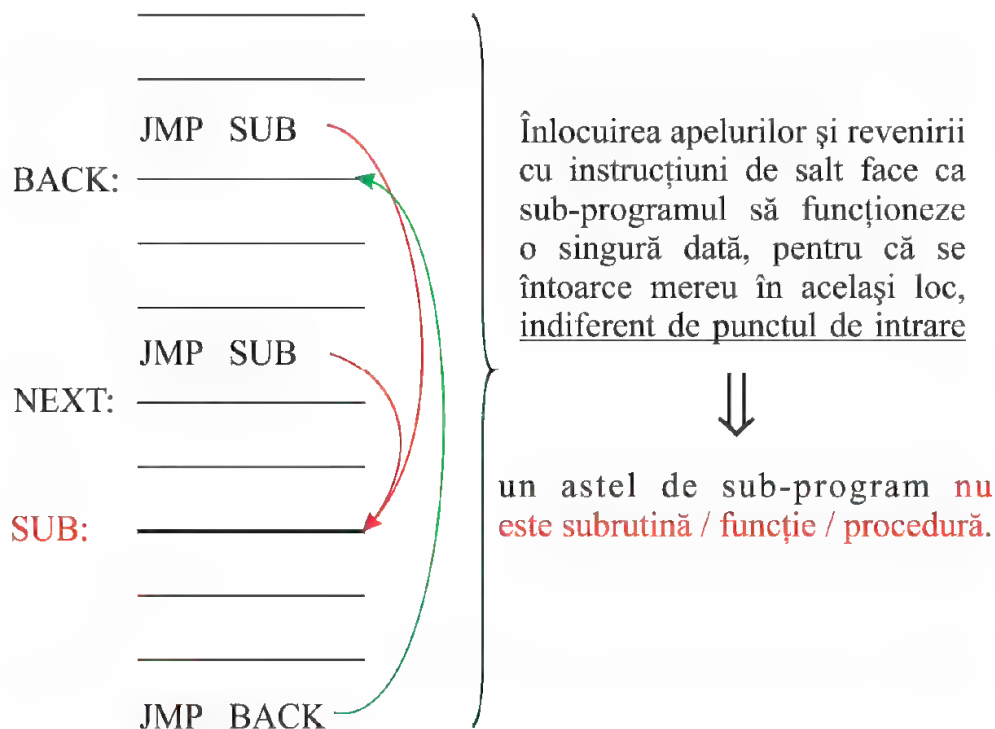
Observații: – La execuția instrucțiunii **RET** valoarea pe 2 octeți aflată în vârful stivei este interpretată implicit ca fiind adresă de revenire ; dacă valoarea a fost alterată, întoarcerea nu se mai face corect:

SUB: _____ } ⇒ diverse instrucțiuni
 _____ }
 _____ }
LXI B,8000H ⇒ **BC = 8000H**
PUSH B ⇒ vârful stivei conține 8000H
RET ⇒ se încarcă în **PC= 8000H**, cu salt într-o zonă necunoscută !

- Dacă subrutina / funcția / procedura este apelată o singură dată, este preferabilă scrierea ei între instrucțiunile programului principal, pentru că apelul și revenirea **consumă timp și spațiu de memorie** !



– Necesitatea stivei la apelul de subrutine rezidă în faptul că trebuie salvate valori diferite ale registrului PC, pentru ca programul să se întoarcă în puncte distincte și nu mereu în același loc:



16.3. Stiva și întreruperile

Definiție: – Se numește întrerupere un eveniment extern care la apariția sa solicită atenția microprocesorului prin activarea unui semnal **hardware** ce produce un apel implicit de subrutină pentru tratare.

– Întreruperea poate apărea între oricare 2 instrucțiuni ale unui program, când este generată o instrucțiune de apel de subrutină prin **hardware**, fără a fi prezentă în program !

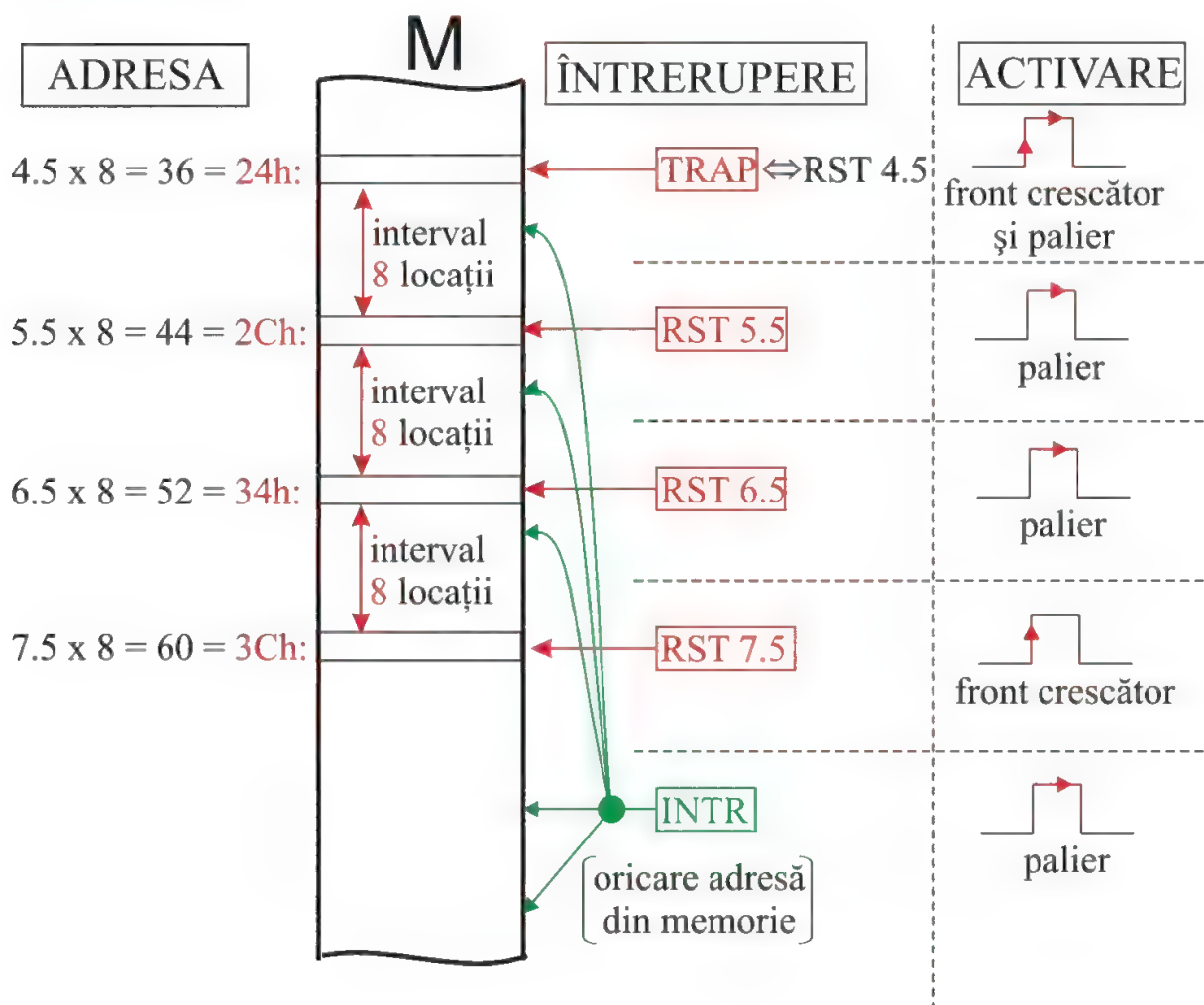
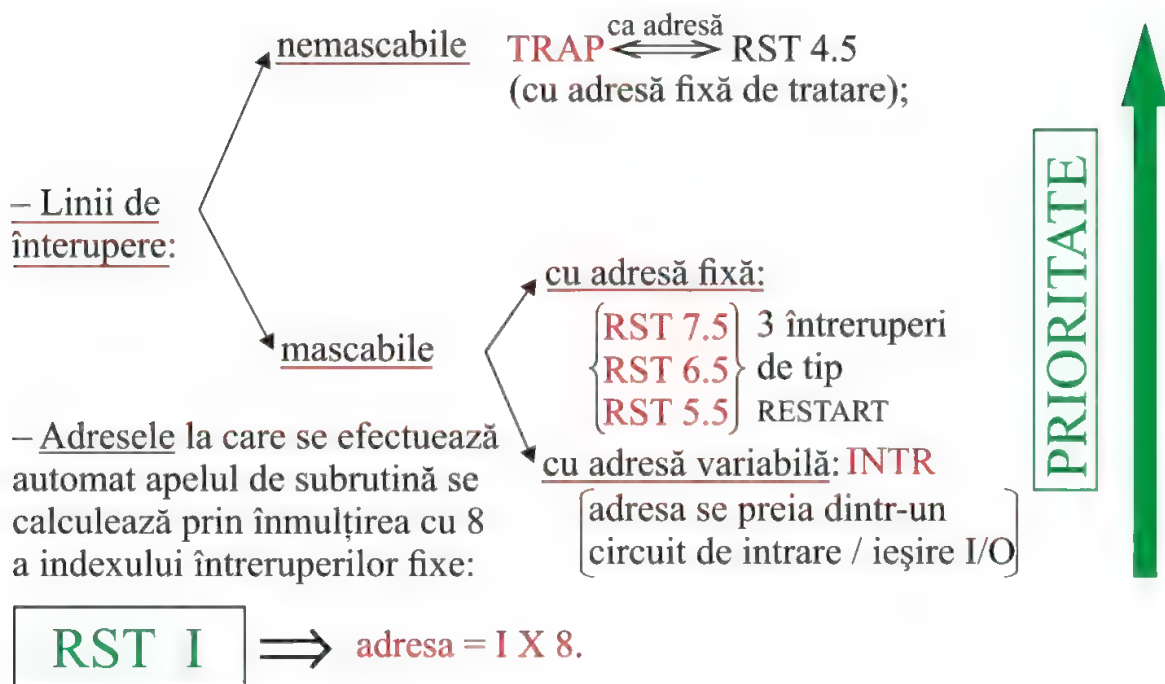
Observație:

– Nu există așa-zise “întreruperi soft” ; acestea sunt doar simple apeluri de subrutine scrise în program.

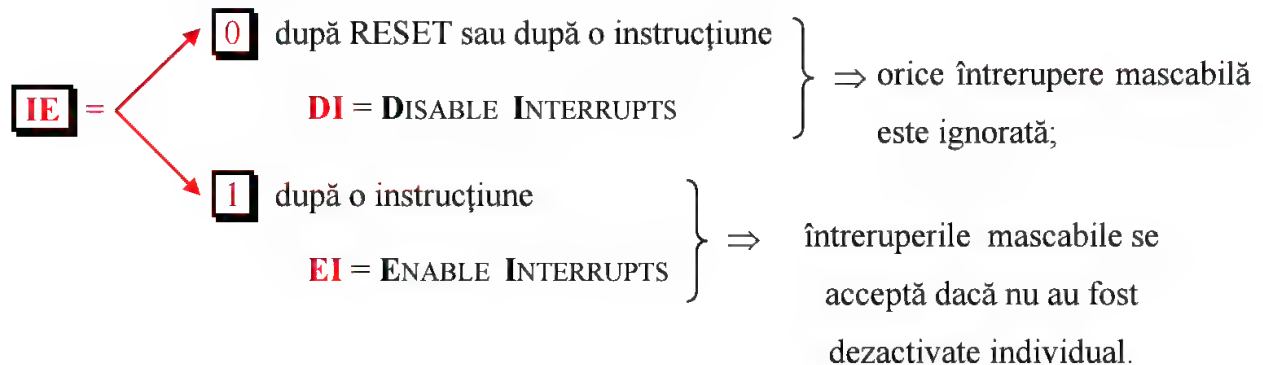
– Întreruperile pot fi:

- ↗ **MASCABILE**, când CPU poate să le dezactiveze global și individual;
- ↘ **NEMASCABILE**, când CPU nu poate să le refuze sau să le dezactiveze.

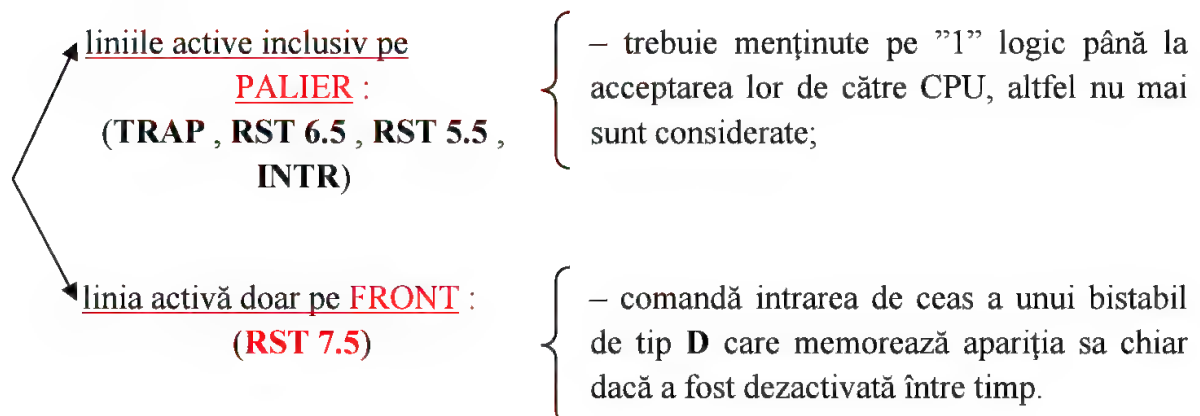
17. Întreruperile microprocesorului 8085



Observație: – Microprocesorul are un bistabil **IE = INTERRUPT ENABLE** de a cărui stare depinde acceptarea tuturor întreruperilor mascabile:



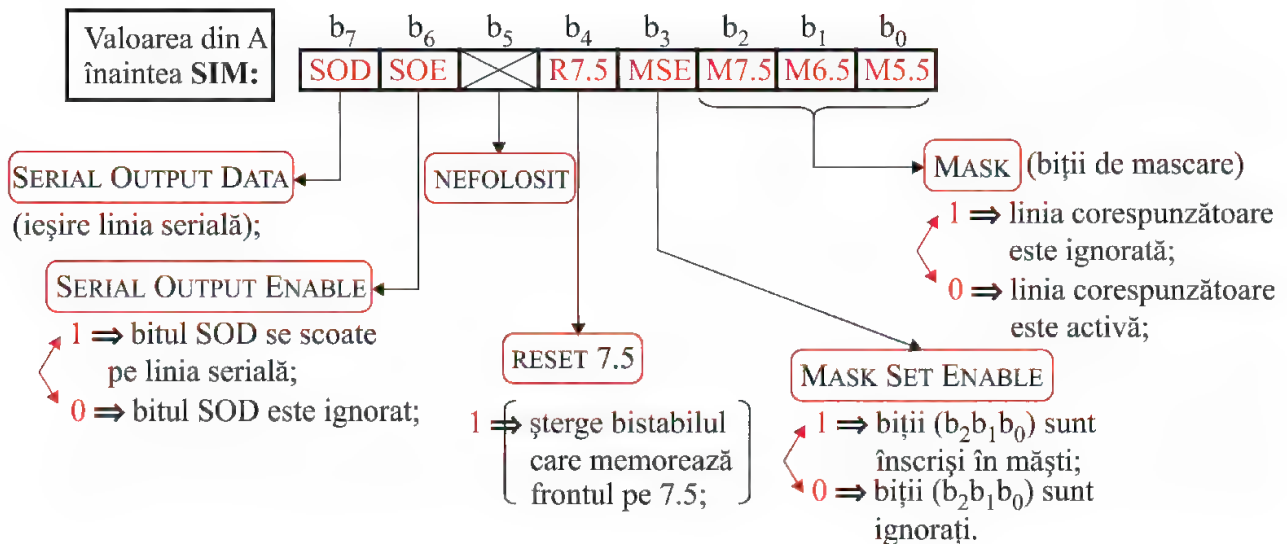
– Acceptarea întreruperilor:



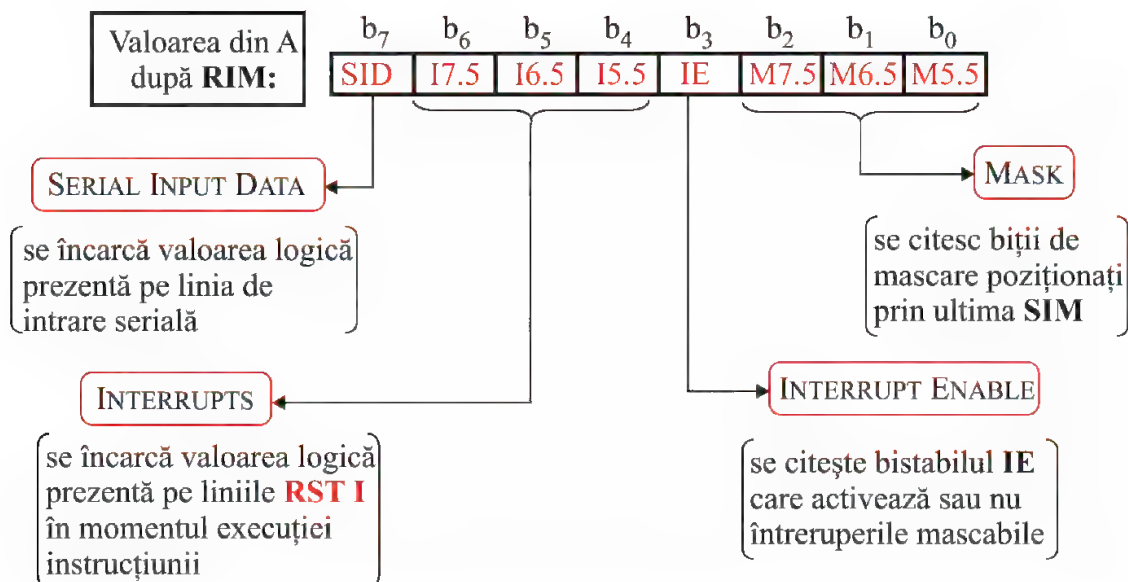
– La acceptare, microprocesorul execută următoarele operații:

- termină de executat instrucțiunea în curs;
- dezactivează întreruperile mascabile executând o operație hardware echivalentă cu instrucțiunea **DI** ;
- salvează în stivă adresa de revenire (valoarea curentă a registrului PC);
- generează hardware o operație echivalentă cu un apel de subrutină la o adresă fixă (sau preia adresa de la un dispozitiv I/O în cazul INTR).

– Înteruperile mascabile cu adresă fixă pot fi dezactivate individual prin **mascare** cu biți “1” logic copiați din acumulator printr-o instrucțiune **SIM = SET INTERRUPT MASK** :



– Microprocesorul **8085** permite citirea stării sistemului de întreruperi prin instrucțiunea **RIM = READ INTERRUPT MASK** , care încarcă în acumulator un octet cu semnificația:



18. Dispozitive IO și configurația sistemului

18.1. Dispozitive de intrare / ieșire

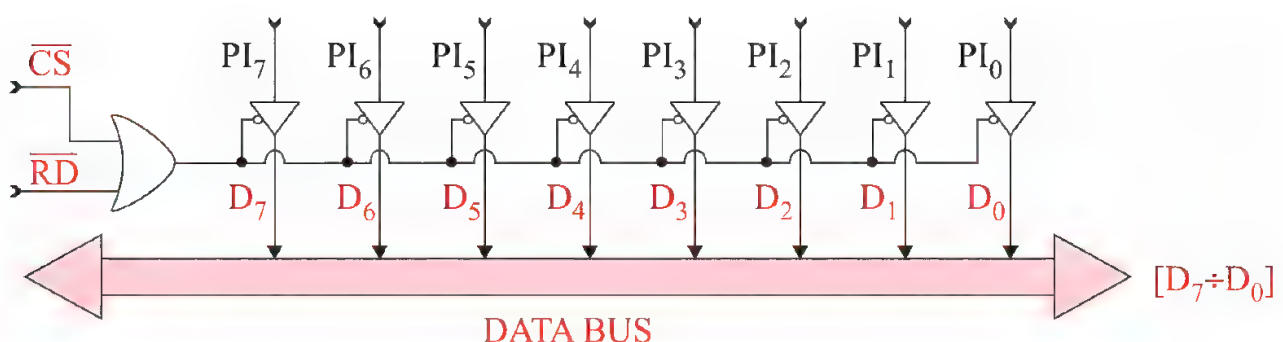
- Sunt structuri hardware combinațional-secvențiale, prin intermediul cărora microprocesorul comunică cu mediul exterior sistemului de calcul (echipamentele periferice).
- Microprocesorul organizează un spațiu de adrese de intrare / ieșire care este complet diferit de spațiul de adrese pentru memorie: distincția între cele două spații de adrese este făcută de semnalul **$IO/\overline{M}$** care este $\begin{cases} "0" & \text{pentru memorie și} \\ "1" & \text{pentru adresă I/O.} \end{cases}$

- Observații:**
- Locațiile din spațiul de I/O se numesc **porturi** ;
 - Microprocesorul **8085** are $2^8 = 256$ de adrese distincte pentru porturi, specificate prin combinația binară a secțiunii inferioare din magistrala de adrese;
 - Secțiunea superioară repetă valoarea depusă pe partea inferioară la accesul de I/O:

$$\text{adresa port} = (A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8)_2 = (A_7A_6A_5A_4A_3A_2A_1A_0)_2$$

- Un dispozitiv de intrare / ieșire poate avea 4 tipuri de porturi: porturi de intrare, porturi de ieșire, porturi de comandă și porturi de stare.

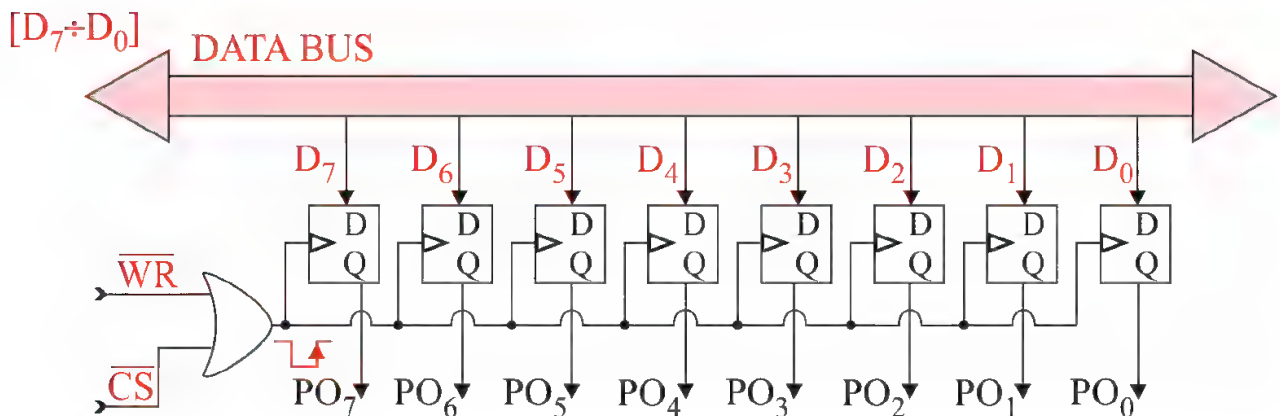
(a.) – porturi de INTRARE : – sunt de obicei fără memorare, alcătuite dintr-un set de 8 “buffer”-e cu 3 stări conectate la magistrala de date;



– pe durata cât ambele semnale $\left\{ \begin{array}{l} \overline{RD} = 0 \text{ (citire)} \\ \text{și} \\ \overline{CS} = 0 \text{ (port selectat)} \end{array} \right\}$ “buffer”-ele sunt conectate

la magistrala de date și CPU poate citi liniile de intrare $[PI_7 \div PI_0]$.

(b.) – porturi de IEȘIRE : – sunt întotdeauna cu memorare, alcătuite dintr-un registru de 8 biți cu intrările conectate la magistrala de date;



– când portul este selectat ($\overline{CS} = 0$) și are loc scrierea ($\overline{WR} = 0$), datele de pe magistrală sunt memorate în bistabile odată cu frontul pozitiv de dezactivare al semnalului ($\overline{WR} : 0 \nearrow 1$).

Observație: – Porturile de ieșire trebuie să fie cu memorare pentru ca informația binară înscrisă să se mențină de la ultima înscrisiere până la următoarea, indiferent ce se vehiculează între timp pe magistrala de date.

(c.) – porturi de COMANDĂ : – sunt porturi de ieșire ale căror linii nu apar efectiv în afara dispozitivului, ci sunt utilizate intern pentru configurare.

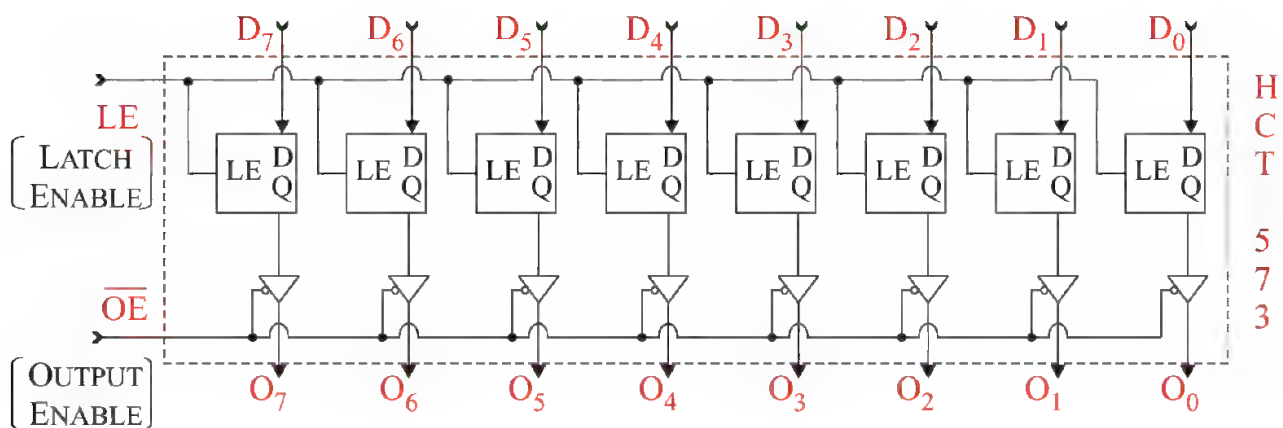
(d.) – porturi de STARE : – sunt porturi de intrare ale căror linii nu provin din afara dispozitivului, ci de la resursele sale interne, permițând CPU să verifice starea.

Observații: – Un port de intrare poate avea aceeași adresă cu un port de ieșire, deoarece informațiile lor sunt vehiculate la momente de timp diferite pe magistrala de date;

– Dacă un port de *comandă* are aceeași adresă cu un port de *stare*, se obișnuiește referirea ansamblului prin **PORT DE CONTROL** al dispozitivului I/O.

18.2. Utilizarea “latch”-ului de 8 biți cu 3 stări HCT 573

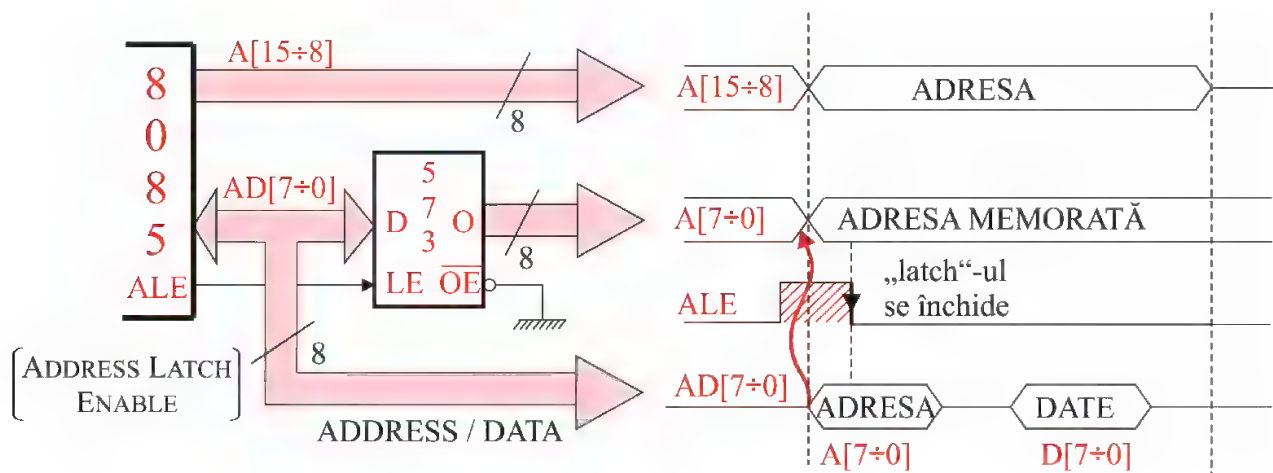
– În interiorul circuitului integrat **HCT 573** se află un set de 8 “latch”-uri cu comandă comună de memorare a intrării (LE), ale căror ieșiri sunt trecute printr-un set de 8 “buffer”-e cu 3 stări, care au comandă comună de deconectare a ieșirii ($\overline{OE}$):



Observație: – Circuitul nu poate fi denumit *registru*, chiar dacă are o structură asemănătoare, pentru că partea de memorare este alcătuită din “latch”-uri de tip **D** (cu **TRANSPARENȚĂ pe nivel**) și nu din bistabile de tip **D** (cu **memorare pe front crescător**).

18.2.1. Utilizarea la memorarea adreselor inferioare:

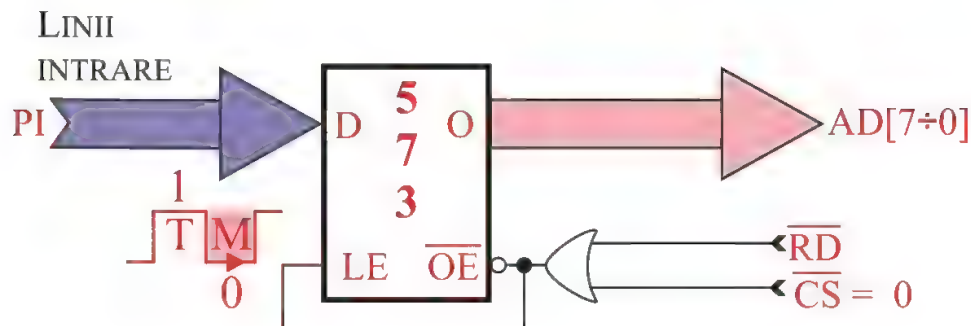
– Se folosește doar *partea de memorare* (cele 8 “latch”-uri), *partea de deconectare* fiind dezactivată (cele 8 “buffer”-e cu 3 stări sunt permanent conectate la ieșiri prin $\overline{OE} = 0$):



- Funcționare:
- înaintea schimbării adreselor, **8085** deschide “latch”-ul prin **ALE = 1** ;
 - apar adresele pe **A[15÷8]** și pe **AD [7÷0]** ;
 - se dezactivează **ALE = 0** și “latch”-ul care era transparent se închide, memorând ultima valoare;
 - adresele inferioare dispar de pe **AD [7÷0]** ;
 - “latch”-ul menține adresele inferioare la ieșire;
 - pe **AD [7÷0]** se vehiculează datele (citire / scriere).

18.2.2. Utilizarea ca port de intrare:

- Se conectează intrarea de memorare împreună cu intrarea de deconectare, ambele fiind comandate simultan de către logica de selecție și semnalul de citire al microprocesorului:

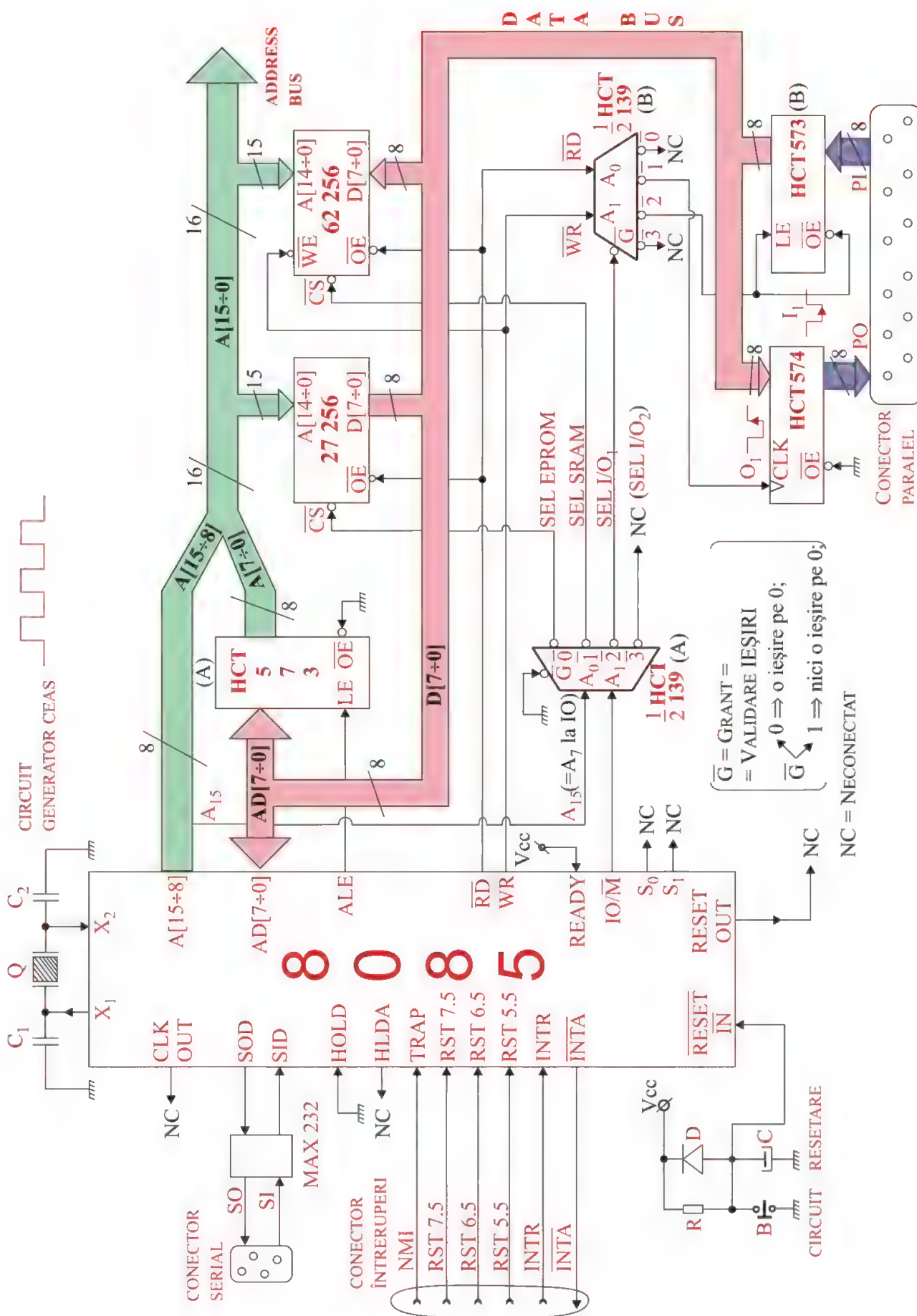


– Când portul **nu** este selectat ($LE = \overline{OE} = 1$), “latch”-urile sunt transparente (T) și preiau semnalele de pe liniile de intrare, dar ieșirile sunt deconectate de la magistrala de date a microprocesorului **AD [7÷0]**;

– Când portul este selectat ($LE = \overline{OE} = 0$), “latch”-urile se închid (M) și memorează ultima valoare prezentă pe liniile de intrare, iar ieșirile se conectează la magistrala de date și transmit valorile memorate către microprocesor;

Observatie: – Avantajul major al configurației cu “latch”-uri față de o configurație care ar utiliza doar “buffer”-e cu 3 stări constă în **menținerea constantă a ieșirilor pe magistrala de date**, pe toată durata ciclului de citire al microprocesorului, indiferent dacă liniile de intrare în port se modifică chiar în acest interval de timp.

18.3. Configuratia unui sistem minimal de calcul



18.3.1. Componentele și rolul lor:

- Anexele microprocesorului 8085 :
- | | | |
|---|---|--|
| <p><u>Semnale nefolosite :</u>
 CLK OUT , RESET OUT ,
 HOLD , HLDA, READY ,
 S1 , S0</p> | { | <p>Q = cristal de quartz pentru stabilizarea generatorului de ceas;
 C₁ , C₂ = condensatoarele generatorului;
 R = rezistor pentru dezactivarea resetării;
 C = condensator electrolitic pentru resetare la conectarea alimentării;
 D = diodă pentru descărcarea condensatorului;
 B = buton de resetare manuală;</p> |
|---|---|--|
- Memorie: { **27 256** = memorie EPROM cu fereastră, capacitate de 32 Ko ;
62 256 = memorie SRAM în capsulă lată, capacitate de 32 Ko ;
- “Latch”-uri:
- | | |
|---|--|
| { | <p>→ HCT 573 = "latch" cu 3 stări pentru memorarea
 (A) adreselor inferioare A[7÷0] ;
 → HCT 573 = "latch" cu 3 stări transparent pentru
 (B) portul de intrare pe 8 biți PI ;</p> |
|---|--|
- Registru **HCT 574** pentru portul de ieșire pe 8 biți **PO** cu “buffer”-ele 3 stări permanent conectate la ieșiri;
- Translator tensiune **MAX 232** pentru interfața serială cu liniile **SID** și **SOD** ale microprocesorului;
- Capsulă **HCT 139** = **2 x DCD 2:4** folosită pentru:
- | | |
|---|--|
| { | <p>→ (A) = decodor pentru logica de selecție memorie și I/O ;
 → (B) = decodor pentru condiționarea cu $\overline{RD}$ și $\overline{WR}$ a semnalului de selecție pentru porturi I/O₁ (în loc de porți OR, care ar mai consuma o capsulă de circuit integrat) .</p> |
|---|--|

18.3.2. Logica de selecție și maparea memorie / intrare – ieșire:

– Logica de selecție are rolul de a selecta un singur circuit de memorie sau de intrare / ieșire, în funcție de semnalul $IO/\overline{M}$ și anumite linii de adresă (în cazul de față numai A_{15}).

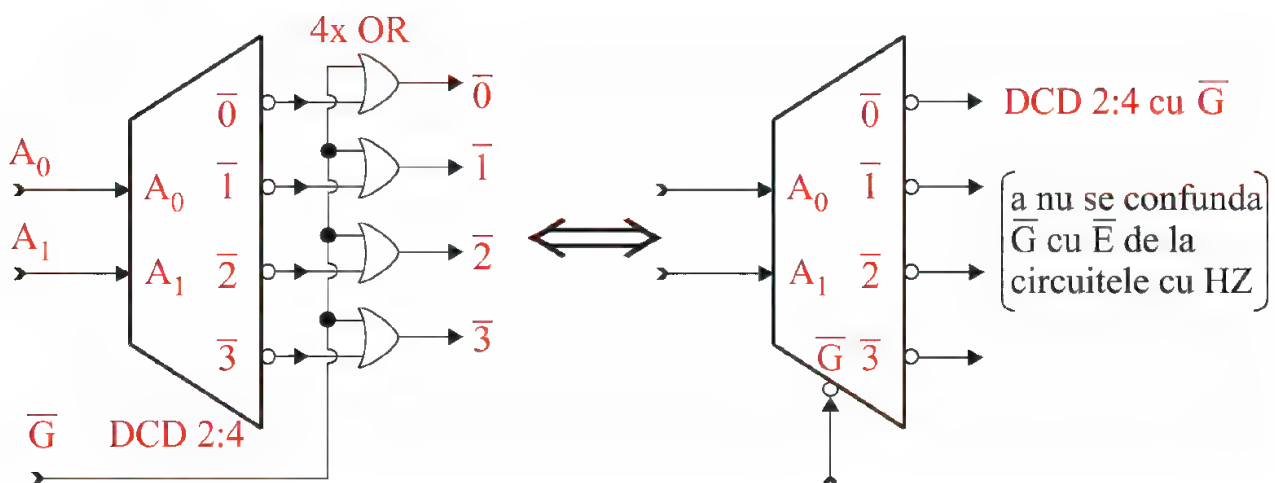
Observații:

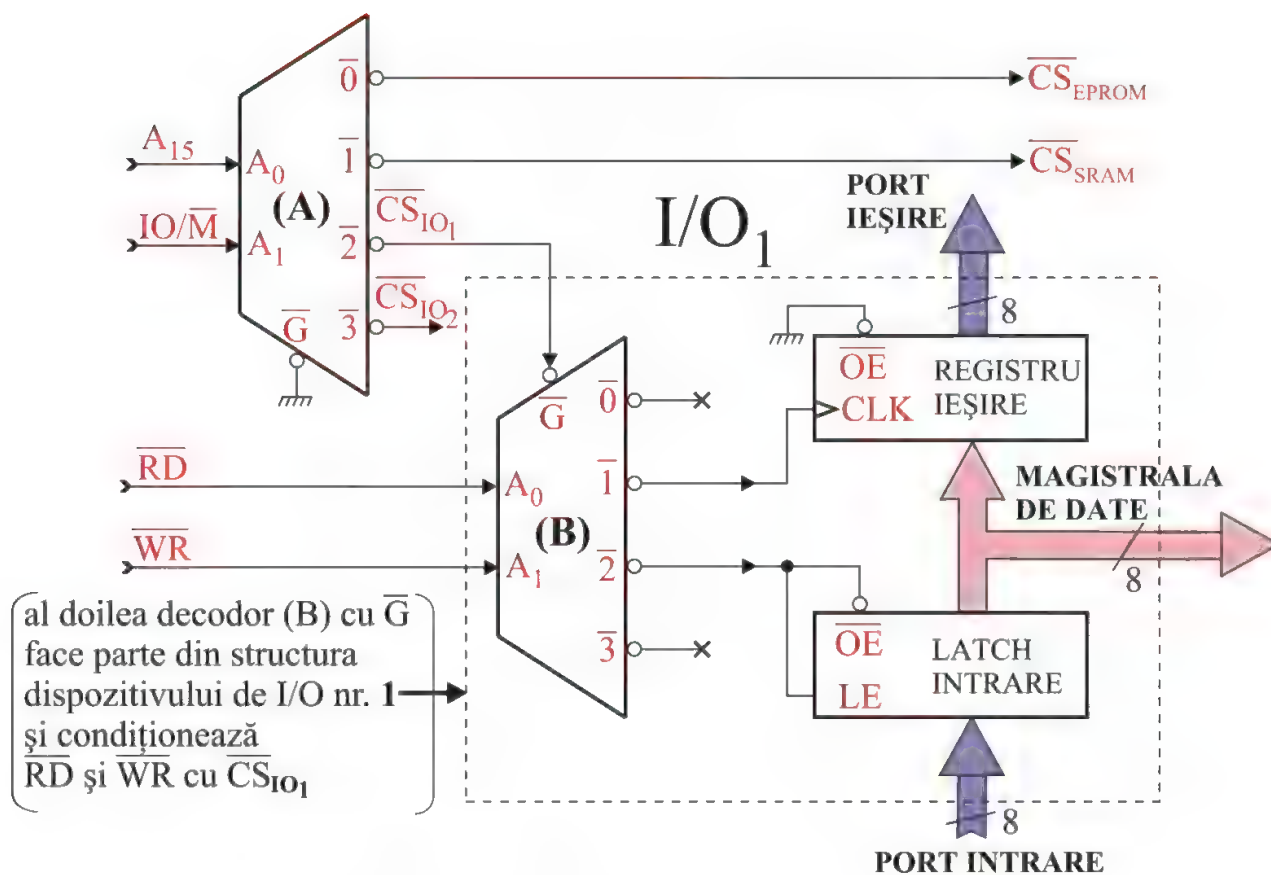
– Decodorul **2:4** cu validare a ieșirilor prin semnalul $\overline{G}$ este echivalent cu un decodor obișnuit cu ieșiri active pe zero care sunt trecute printr-un set de 4 porți OR pentru validare:

$$\left\{ \begin{array}{l} \overline{G} = 0 \Rightarrow \text{ieșirea activă trece mai departe;} \\ \overline{G} = 1 \Rightarrow \text{nici o ieșire nu trece mai departe,} \\ \text{toate sunt "1" logic (inactive);} \end{array} \right.$$

– Facilitatea de validare a ieșirii active nu este folosită la secțiunea de selectare a porturilor ($\overline{G}$ este permanent la 0), dar la secțiunea de condiționare cu semnalele de citire / scriere pentru portul I/O_1 este folosită doar partea de validare și nu decodorul propriu-zis;

– Implementarea logicii de selecție prin decodare reduce numărul de capsule de circuite integrate folosite și complexitatea cablajului de interconectare.



– Funcționarea logicii de selecție:**DECODOR SELECȚIE:**

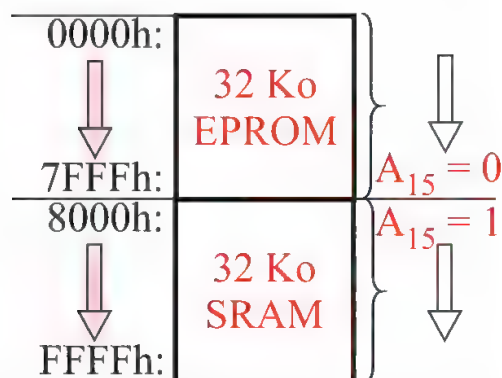
$IO/\overline{M}$	A_{15}	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	SELECȚIE	
0	0	0	1	1	1	EPROM	acces MEMORIE
0	1	1	0	1	1	SRAM	
1	0	1	1	0	1	I/O ₁	acces I/O
1	1	1	1	1	0	I/O ₂	

← nefolosit, pentru extinderi ulterioare

DECODOR CONDIȚIONARE:

$\overline{G}$	$\overline{WR}$	$\overline{RD}$	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	CONDIȚIONARE	
0	0	0	IMPOSIBIL					acces IO ₁ cu: $\boxed{IO/\overline{M} = 1}$; și $\boxed{A_{15} = 0}$ (= A ₇ la IO).
0	0	1	1	0	1	1	O ₁ (WRITE)	
0	1	0	1	1	0	1	I ₁ (READ)	
0	1	1	1	1	1	0	INACTIV	
1	–	–	1	1	1	1	alte dispozitive accesate	

– **Maparea memoriei și dispozitivelor I/O:** – Se construiește câte o hartă pentru spațiul de adrese al memoriei și spațiul de adrese al porturilor de intrare-ieșire:



- Spațiul de adrese al memoriei de **64 Ko** este împărțit în 2 secțiuni egale; în fiecare secțiune se accesează câte o locație distinctă din EPROM / SRAM.

- Spațiul de adrese I/O de **256 octeți** este utilizat pe jumătate, cu acces separat pentru citire și scriere:



- la citire, pentru **orice** adresă între **00h÷7Fh** se citește același „latch” de intrare;

- la scriere, pentru **orice** adresă între **00h÷7Fh** se scrie același registru de ieșire;

Observatie: – O astfel de selecție **REDUNDANTĂ** (pentru adrese diferite se selectează același port) este specifică dispozitivelor de I/O, care au mult mai puține locații decât spațiul de adrese, spre deosebire de memorie.

19. Probleme și subiecte teoretice propuse

1. Să se reprezinte următoarele numere fracționare fără semn în bazele 2, 4, 8 și 16:
 - a) $N = 206.17578125|_{10}$;
 - b) $N = 183.53515625|_{10}$.

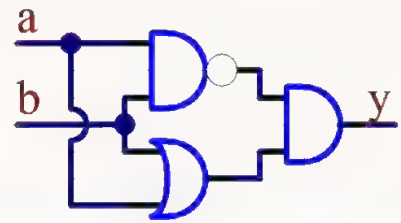
2. Să se reprezinte numărul întreg zecimal $N = -323$ în mărime și semn, în cod complementar față de 1 și în cod complementar față de 2, folosind în total 10 biți.

3. Se dau două numere de 8 biți fără semn, notate A și B. Să se scrie toate operațiile necesare pentru a calcula un număr C de forma:
 - a) $C = (a_5 a_4 b_7 b_6 01 \bar{a}_0 \bar{b}_1)|_2$;
 - b) $C = (a_4 a_3 b_2 b_1 10 \bar{b}_7 \bar{a}_7)|_2$;
 - c) $C = (b_3 b_2 a_6 a_5 10 \bar{b}_0 \bar{a}_1)|_2$;
 - d) $C = (a_6 a_5 b_4 b_3 01 \bar{a}_7 \bar{b}_7)|_2$;

4. Se consideră două numere de 8 biți fără semn A și B. Folosind doar cele 4 operații de rotație și bitul auxiliar Cy (fără alte variabile auxiliare și fără operații logice), să se calculeze din octetul $A = (a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0)|_2$ un octet B de forma:
 - a) $B = (a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7)|_2$ (cu ordinea biților inversată);
 - b) $B = (a_0 a_2 a_4 a_6 a_1 a_3 a_5 a_7)|_2$ (cu ordinea biților amestecată).

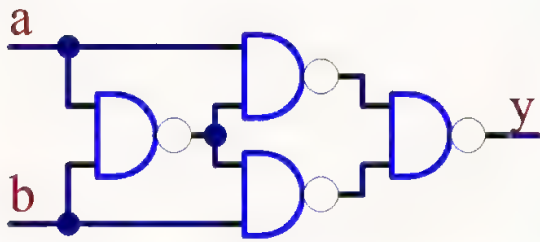
5. Se consideră trei numere de 8 biți fără semn A, B și C. Folosind doar cele 4 operații de rotație și bitul auxiliar Cy (fără alte variabile auxiliare și fără operații logice), să se calculeze combinația dintre A și B de forma:
 - a) $C = (a_0 b_0 a_1 b_1 a_2 b_2 a_3 b_3)|_2$;
 - b) $C = (a_0 b_4 a_1 b_5 a_2 b_6 a_3 b_7)|_2$.

6. Să se analizeze schema combinațională cu porți din dreapta și să se aducă la forma cea mai simplă, folosind doar algebra logică.

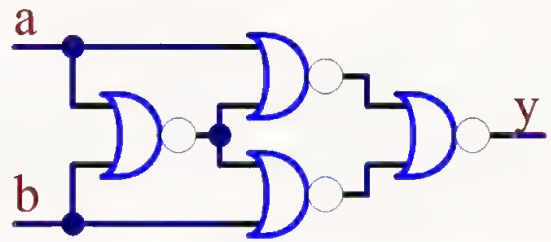


7. Să se calculeze expresia matematică cea mai simplă a funcțiilor logice $y(a,b)$ din figurile de mai jos, aplicând axiomele și teoremele algebrei logice:

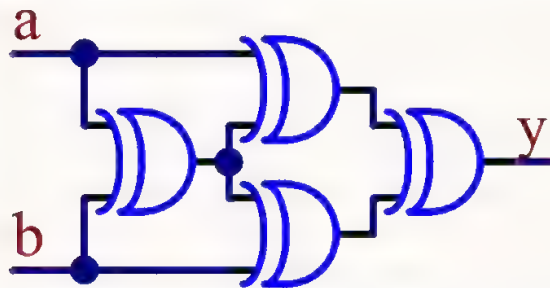
a)



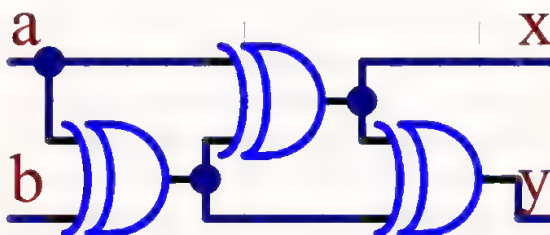
b)



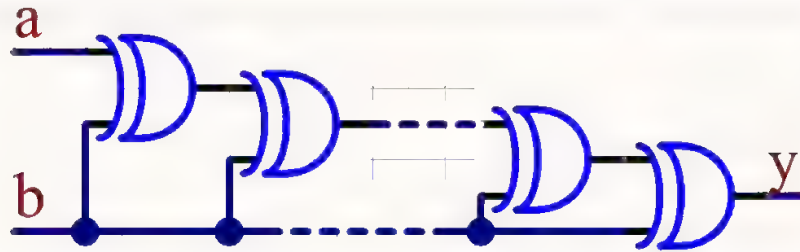
8. Să se determine funcția logică $y(a,b)$ implementată prin schema următoare și să se deseneze structura echivalentă simplificată:



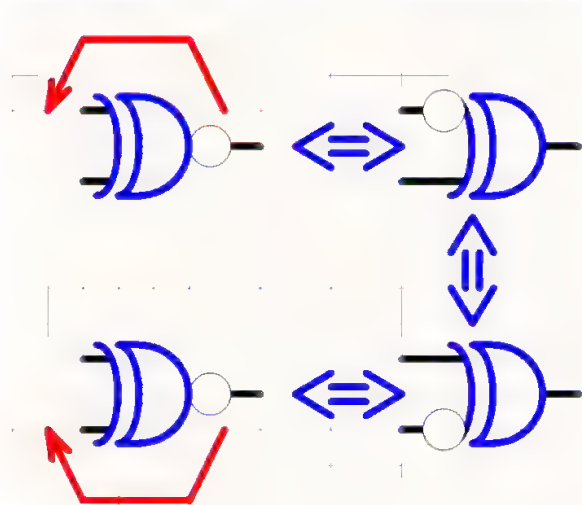
9. Folosind caracteristicile speciale ale porții XOR, să se determine ieșirile logice x și y în figura de mai jos. Ce s-ar obține după repetarea în cascadă a structurii logice (intrările următoare conectate la ieșirile curente) de un număr par sau impar de ori?



10. Un număr mare de porți XOR sunt conectate în cascadă, după schema din figură. Folosind proprietățile speciale ale porții XOR, să se determine funcția logică $y(a,b)$ implementată cu un număr total de **10** și apoi de **11** porți:



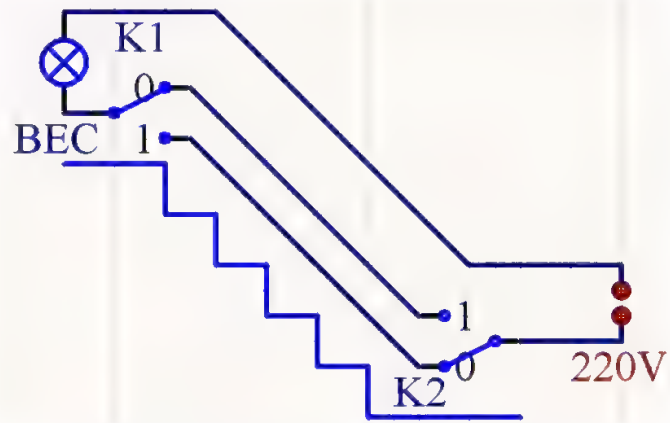
11. Folosind tabele de adevăr și apoi algebra logică, să se verifice dacă negarea de la ieșirea unei porți XOR se poate transfera numai la una din intrările sale, fără a modifica poarta, spre deosebire de teoremele lui De Morgan (unde negarea de la ieșire se transferă la ambele intrări simultan, schimbând totodată poarta în complementara ei):



12. Schema electrică de comandă independentă pentru un bec de iluminare a unei scări este prezentată în figura care urmează și presupune utilizarea a două comutatoare cu câte 2 poziții fiecare, funcționând astfel:

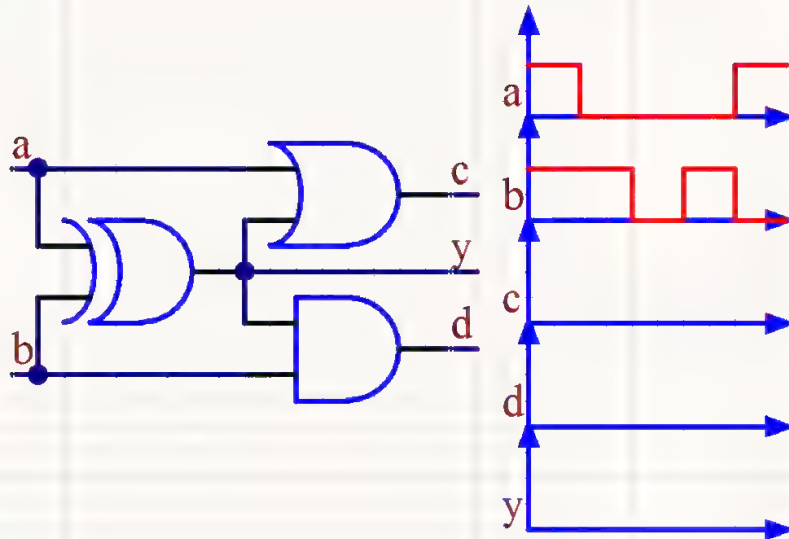
- Inițial becul este stins;
- Când o persoană trebuie să urce scara, acționează comutatorul K2 ca să aprindă becul, urcă scara, apoi acționează comutatorul K1 ca să stingă becul;
- Oricâte alte persoane care urcă scara pot proceda în același mod;
- Când o persoană trebuie să coboare scara, acționează comutatorul K1 ca să aprindă becul, coboară scara, apoi acționează comutatorul K2 ca să stingă becul;
- Oricâte alte persoane care coboară scara pot proceda în același mod.

Să se determine logica după care se aprinde becul în funcție de starea comutatoarelor și să se deseneze o schemă echivalentă de comandă cu porți logice:

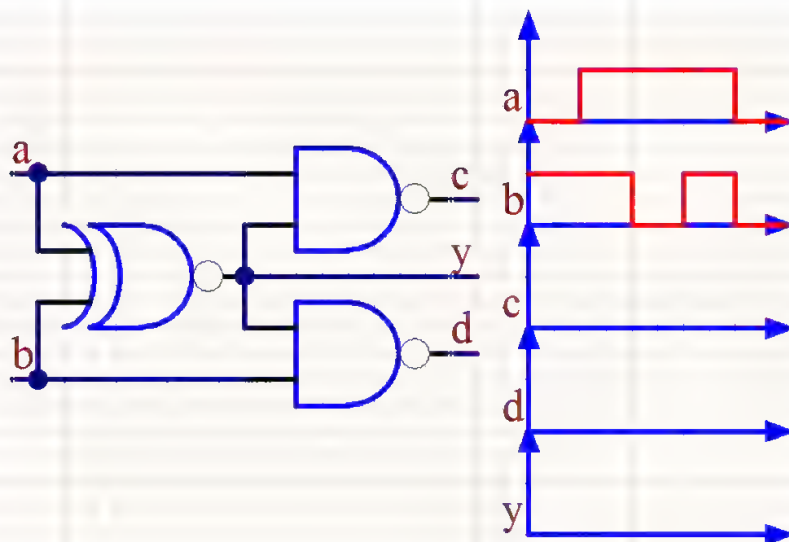


13. Să se completeze diagrama de timp a logicii combinaționale din figurile următoare cu semnalele lipsă, explicând modul de rezolvare:

a)

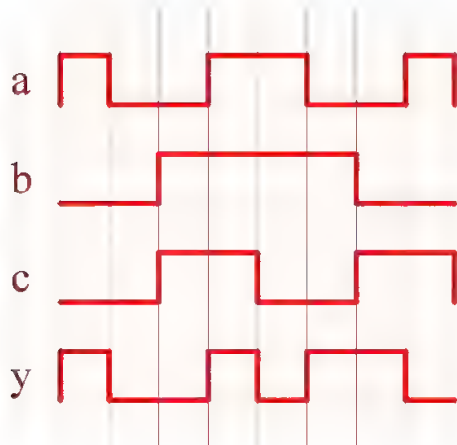


b)

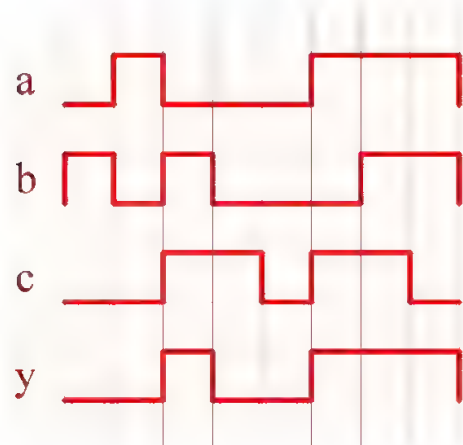


14. Într-un proces de “*reverse engineering*”, o structură logică combinațională necunoscută este analizată prin aplicarea unor semnale la cele 3 intrări ***a***, ***b***, ***c*** și urmărirea semnalului rezultat la ieșirea ***y***, obținând astfel o diagramă de timp. Să se determine o schemă logică cât mai simplă care să poată înlocui structura analizată în aplicații, pentru fiecare din cazurile următoare:

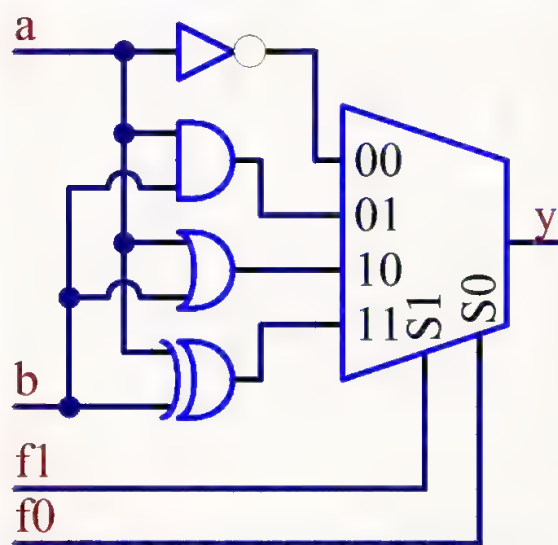
a)



b)

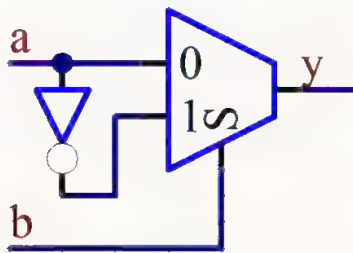


15. O secțiune de un rang dintr-o unitate aritmetico-logică este implementată conform schemei din figura următoare. Să se analizeze funcționarea acesteia, apoi să se găsească funcția de ieșire sub formă de tabel compact de adevăr și sub formă de relație booleană:

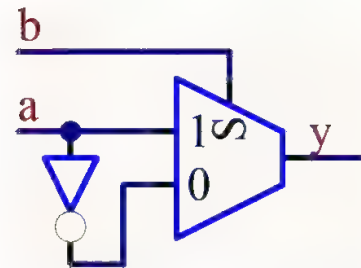


16. Să se explice în cuvinte funcționarea schemelor de mai jos, apoi să se determine funcțiile logice implementate și să se deseneze structuri echivalente care folosesc doar porți logice:

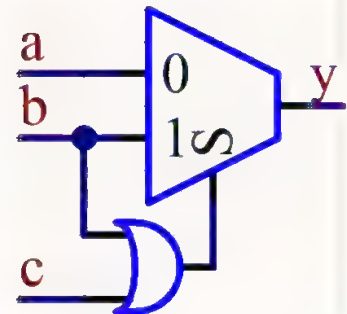
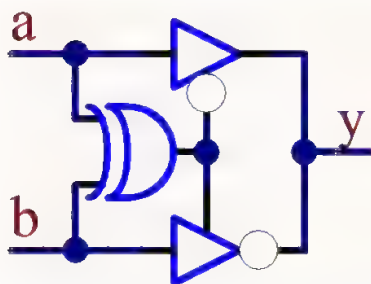
a)



b)

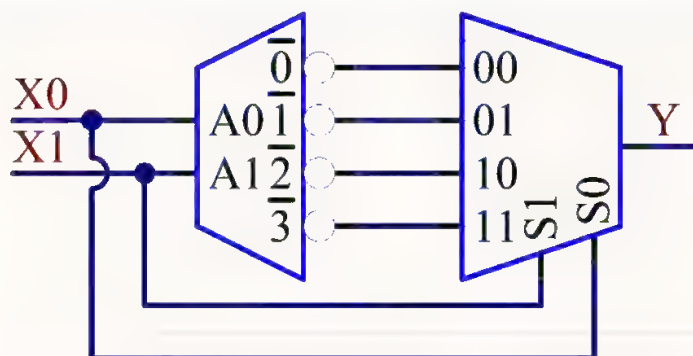


17. Să se determine funcția logică $y(a,b)$ implementată prin schema din dreapta și să se deseneze pentru ea o structură echivalentă cu porți logice.



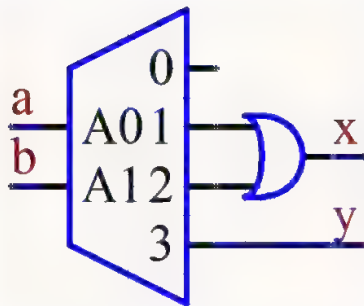
18. Dacă nu apare scurt-circuit la ieșirile conectate împreună, să se determine funcția logică $y(a,b)$ implementată prin schema din stânga.

19. Să se determine funcția logică implementată prin schema următoare, explicând cum funcționează ea și să se descrie aceeași funcție folosind doar porți logice:

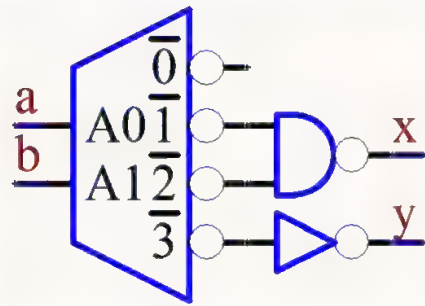


20. Să se analizeze funcționarea schemelor următoare, apoi să se implementeze aceeași funcție doar cu porți logice și să se găsească blocul combinațional standard echivalent:

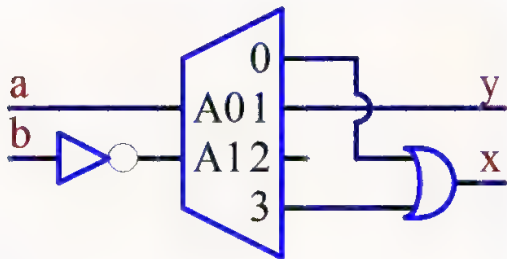
a)



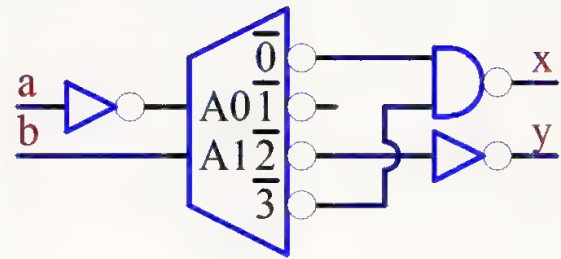
b)



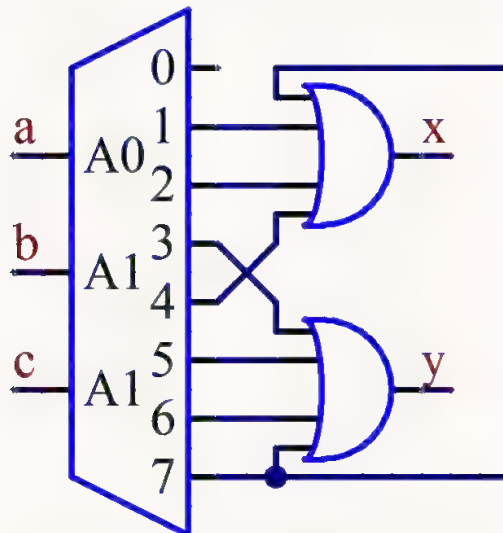
c)



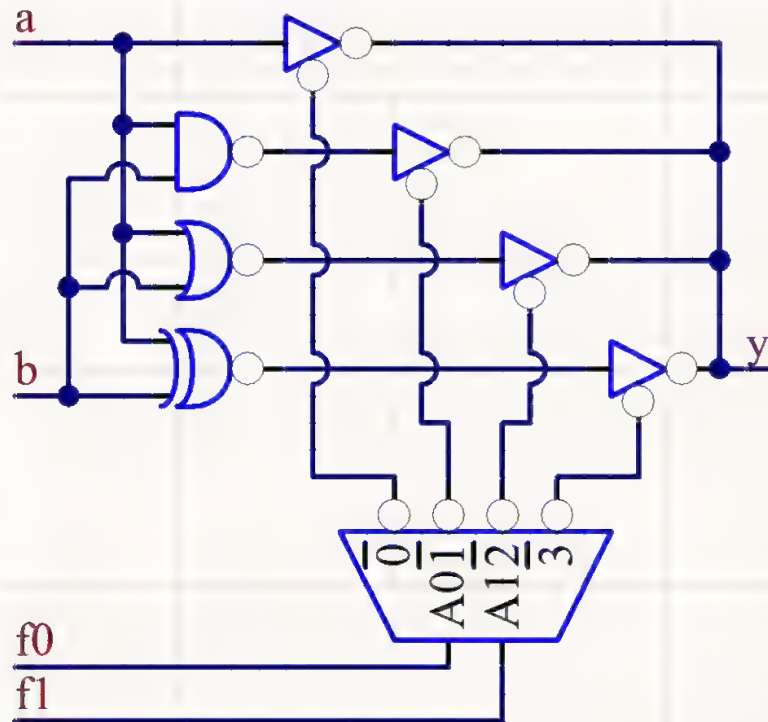
d)



e)

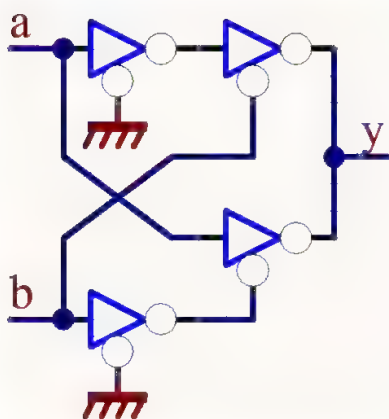


21. Să se explice de ce schema care urmează nu produce scurt-circuit la ieșirile componentelor analizând funcționarea ei în cuvinte, apoi să se găsească funcția de ieșire sub formă de tabel compact de adevăr și sub formă de relație booleană:

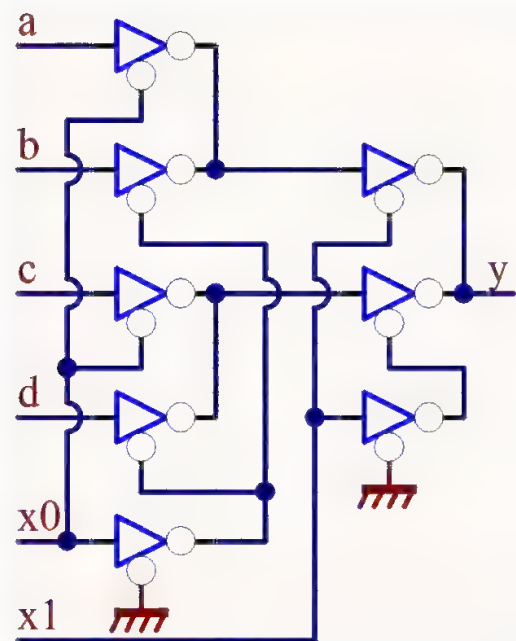


22. Să se explice de ce în schemele următoare nu se produc scurt-circuite la ieșirile conectate împreună, apoi să se determine funcția logică implementată și să se găsească blocul combinațional echivalent ca funcționare:

a)

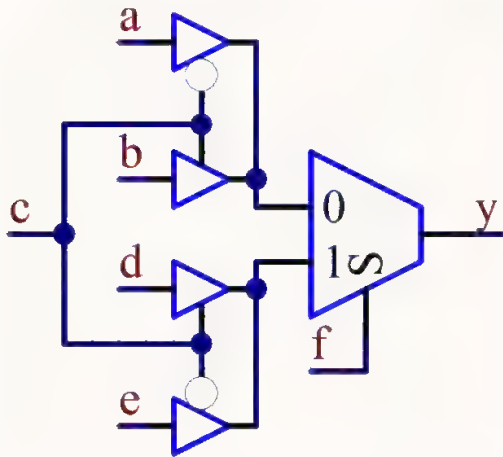


b)

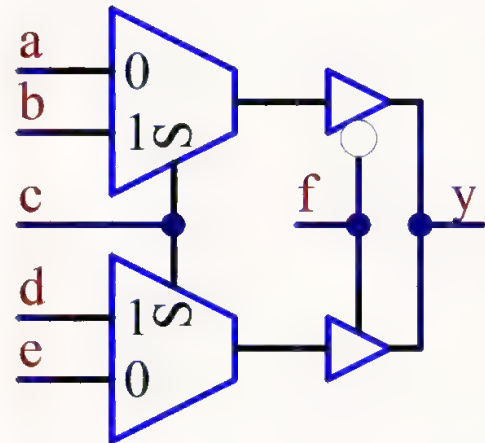


23. Să se verifice dacă în schemele următoare se produc sau nu scurt-circuite pe ieșirile conectate împreună, apoi să se explice cum funcționează și să se găsească un bloc combinațional echivalent cu aceeași funcție logică:

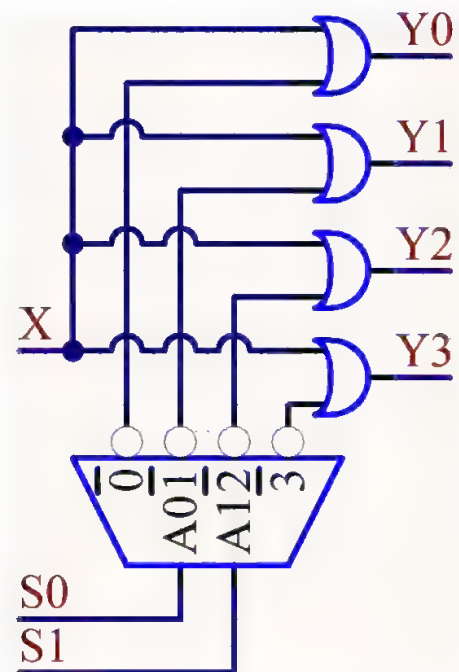
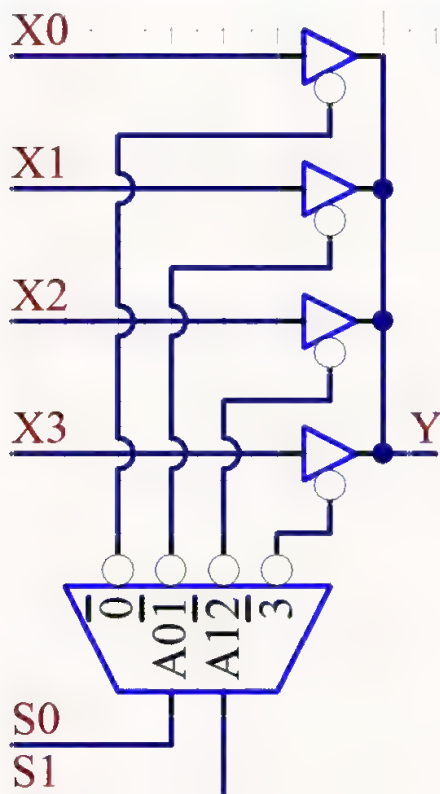
a)



b)



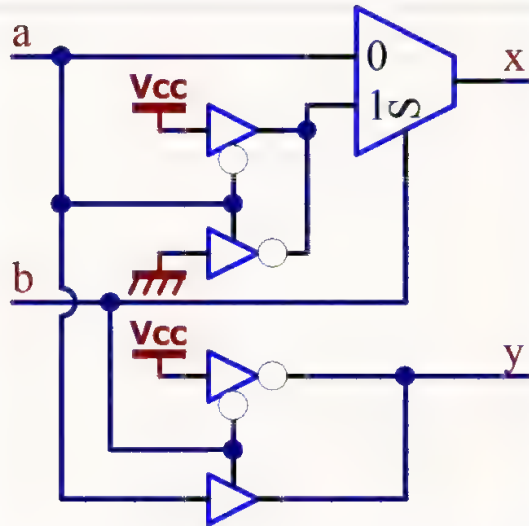
24. Să se explice funcționarea schemei logice din dreapta, apoi să se identifice blocul combinațional standard care implementează aceeași logică și să se deseneze simbolul său echivalent.



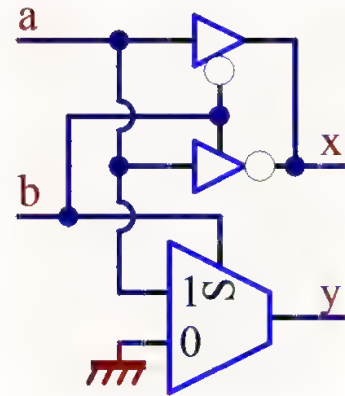
25. Să se explice funcționarea schemei din stânga, analizând dacă există posibilitatea de scurt-circuit la ieșirile conectate împreună, apoi să se identifice și să se deseneze blocul combinațional standard care implementează aceeași logică.

26. Să se explice de ce în schemele de mai jos nu se produc scurt-circuite la ieșirile componentelor, apoi să se implementeze aceeași funcție logică cu porți obișnuite și să se găsească blocul combinațional standard echivalent:

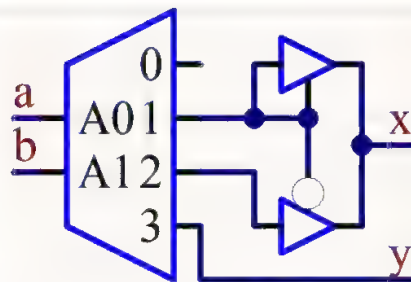
a)



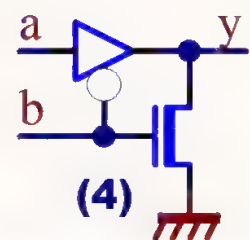
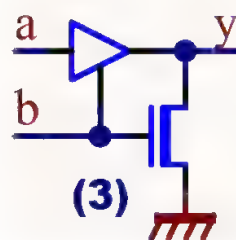
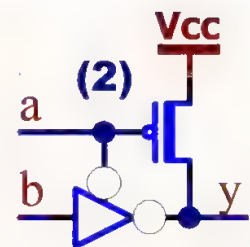
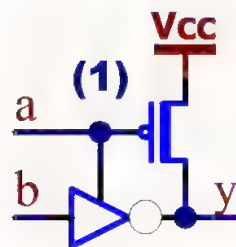
b)



c)

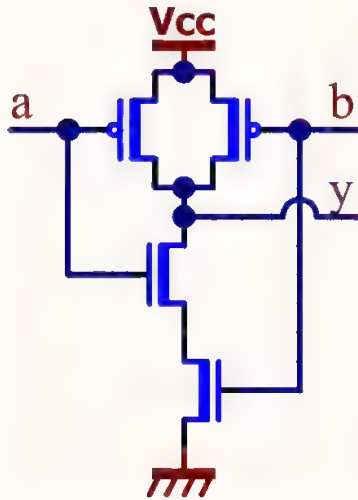


27. Să se analizeze prin tabele și explicații detaliate în cuvinte dacă schemele (1), (2) și (3), (4) din dreapta produc scurt-circuit la ieșire sau implementează o funcție logică utilă, care se va descrie în acest caz prin formulă și schemă echivalentă cu porți obișnuite.

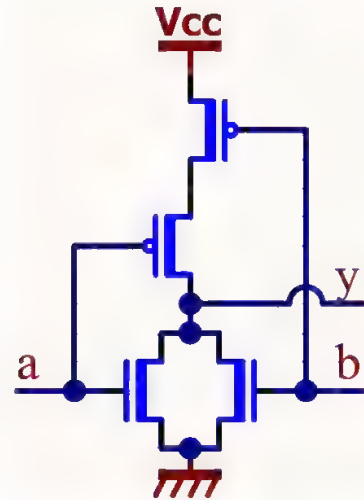


28. Două tranzistoare de tip N și două tranzistoare de tip P se conectează ca în figurile din continuare. Pentru fiecare schemă, să se explice funcționarea ansamblului de tranzistoare și să se determine structura lui echivalentă cu porți logice:

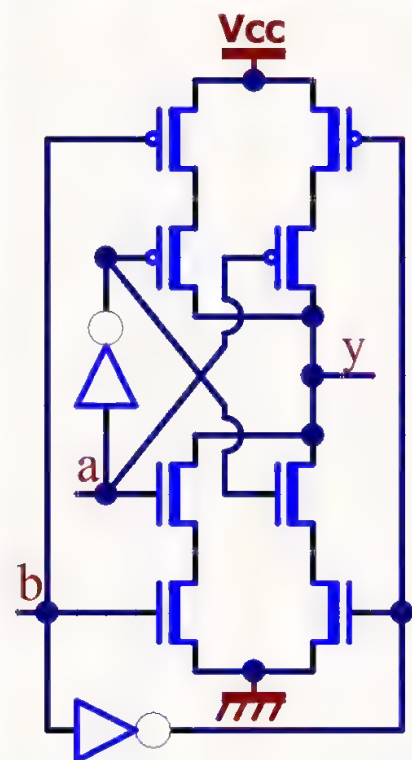
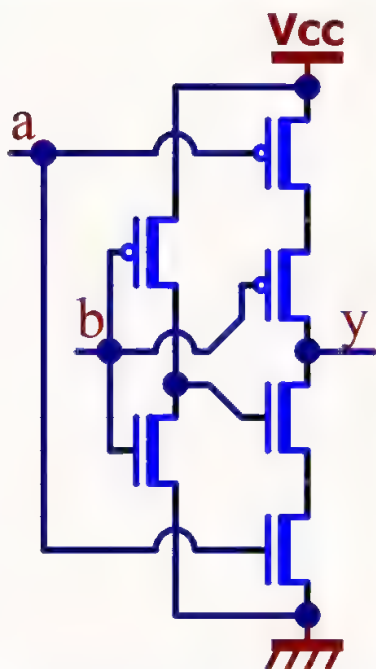
a)



b)

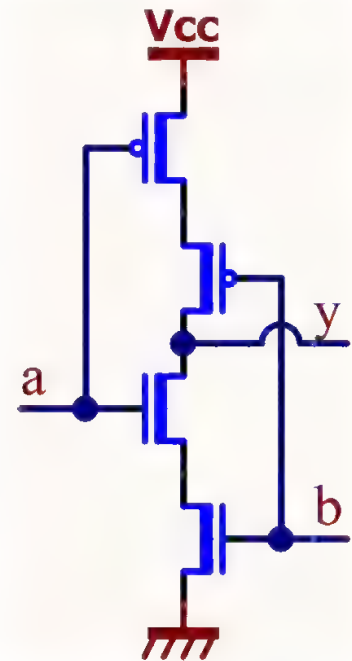


29. Să se analizeze funcționarea schemei combinaționale din dreapta, folosind un tabel de adevăr în care se va nota starea fiecărui tranzistor ($C = \text{"conduce"}$ sau $NC = \text{"nu conduce"}$); dacă nu se produce scurt-circuit între masă și alimentare prin conducția tranzistoarelor, să se determine funcția logică de ieșire.



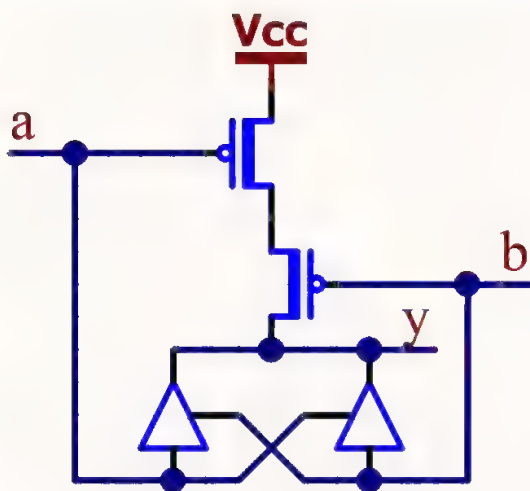
30. Să se explice funcționarea ansamblului de tranzistoare conectate după schema din stânga și să se determine structura logică pe care o implementează.

31. Să se analizeze prin tabele și explicații detaliate în cuvinte stările logice pe care le poate lua ieșirea y pe schema combinațională din dreapta, dacă se produce scurt-circuit și dacă se poate înlocui cu porți sau cu un bloc echivalent standard.

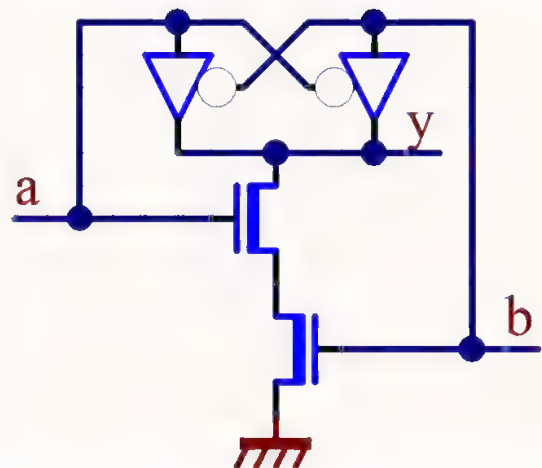


32. Să se analizeze prin tabele și explicații în cuvinte dacă în schemele din figurile următoare se produce scurt-circuit la ieșire sau ele implementează o funcție logică utilă, care se va descrie în acest caz prin schemă echivalentă cu porți sau bloc combinațional standard:

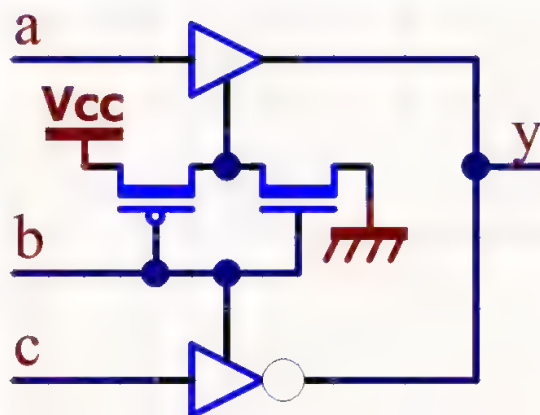
a)



b)

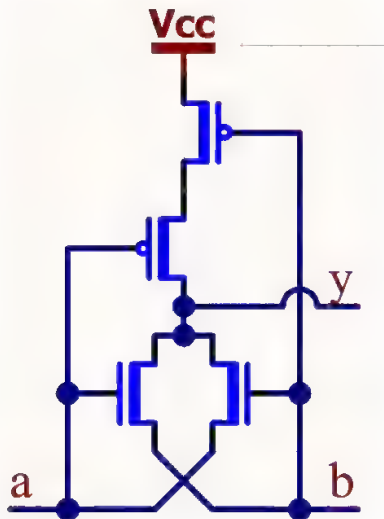


c)

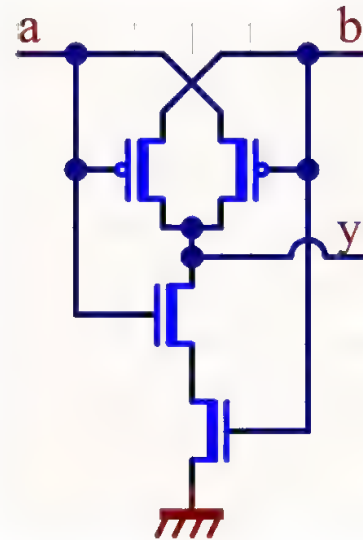


33. Să se explice funcționarea schemelor următoare cu tranzistoare, considerând valorile logice care pot trece prin canalul de conducție în funcție de cele aplicate pe grila de comandă și să se verifice dacă acestea pot sau nu implementa direct o funcție logică utilă:

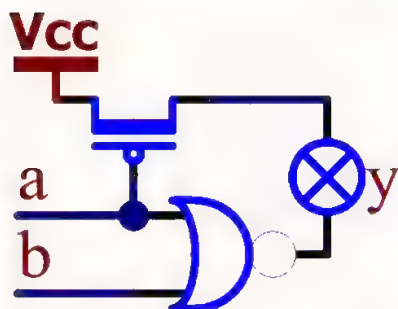
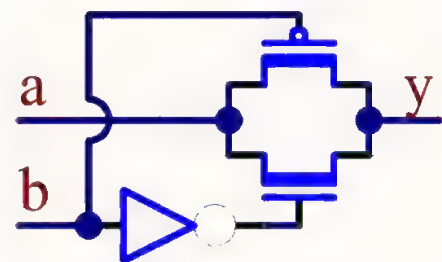
a)



b)



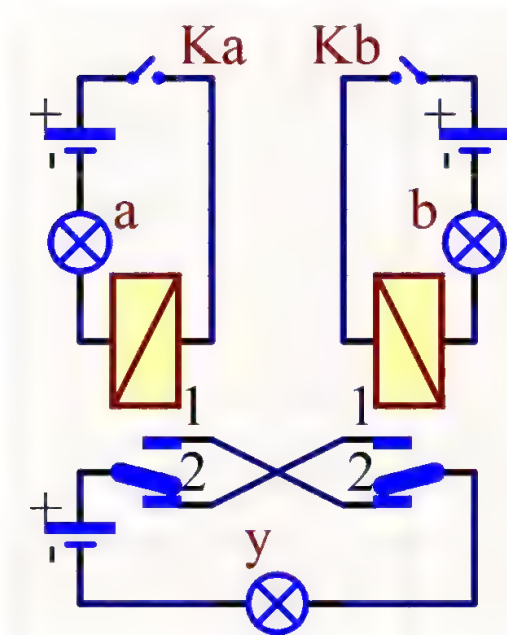
34. Analizând condițiile de conducție ale tranzistoarelor și valorile logice care pot trece prin ele, să se explice funcționarea schemei din dreapta, apoi să se găsească structura combinațională cu același rol.



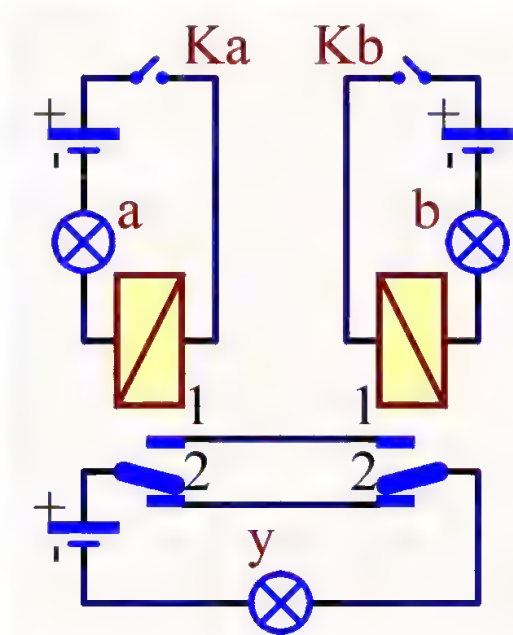
35. Considerând condițiile necesare trecerii curentului electric printr-un circuit, să se explice funcționarea schemei din stânga, când luminează becul și care este formula booleană pentru funcția logică $y = \text{"bec aprins"}$.

36. Contactele comandate de bobinele a două relee electro-magnetice se conectează ca în figurile următoare. Considerând că becurile indică starea logică a circuitelor electrice, să se explice funcționarea structurilor și să se găsească schema lor echivalentă cu porți logice:

a)

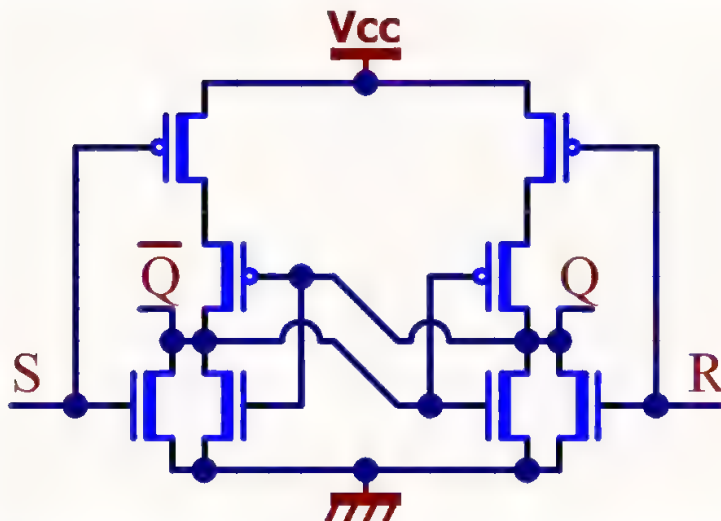
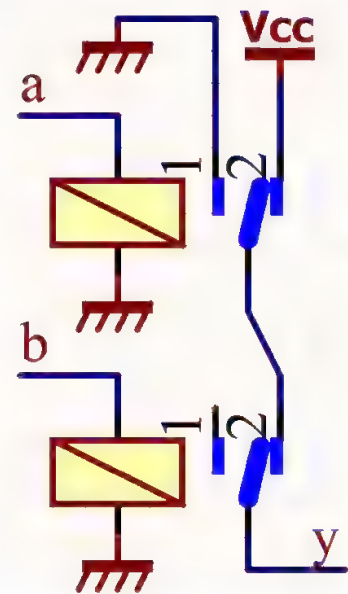


b)

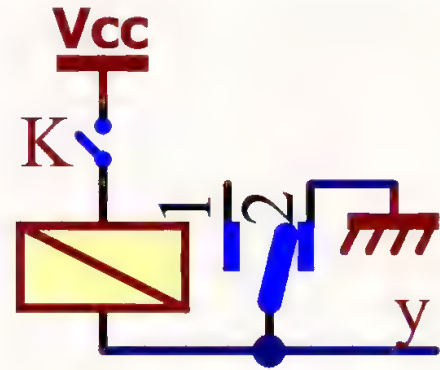


37. Contactele comandate de bobinele a două relee electro-magnetice se conectează ca în figura din dreapta. Să se explice funcționarea ansamblului și să se determine structura logică pe care o implementează.

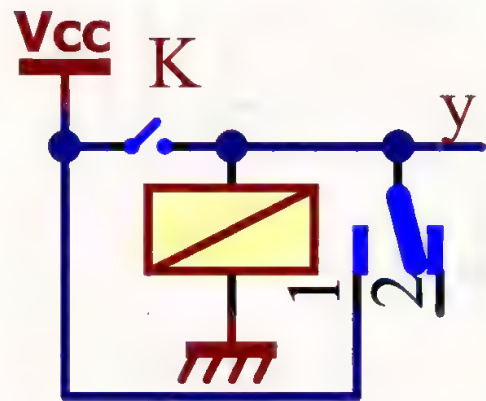
38. Să se analizeze funcționarea celulei secvențiale RS cu tranzistoare de mai jos, considerând aceeași succesiune de comenzi ca la celula RS cu porți NOR și notând starea tuturor tranzistoarelor în fiecare etapă:



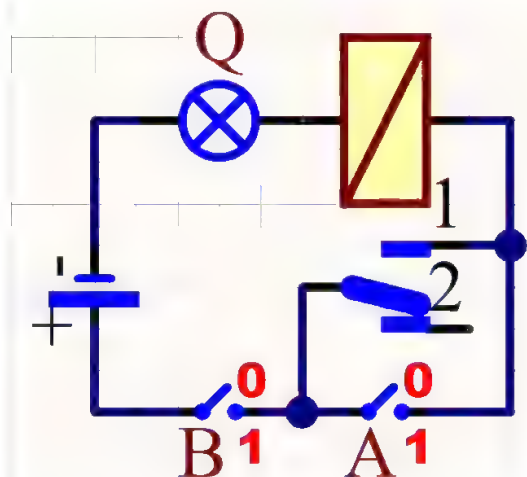
- 39.** Considerând că bobina unui releu permite trecerea valorilor logice prin ea fără a produce scurt-circuit (adică nu există nici un contact direct, dar nici o deconectare totală de tip “High Z” între contactele sale), să se explice funcționarea schemei alăturată și să se deseneze diagrama de timp pentru ieșirea logică y . Se va analiza ce se întâmplă *înainte* de închiderea comutatorului K, *pe durata* cât acesta rămâne închis și *după* ce K a fost din nou deschis. L



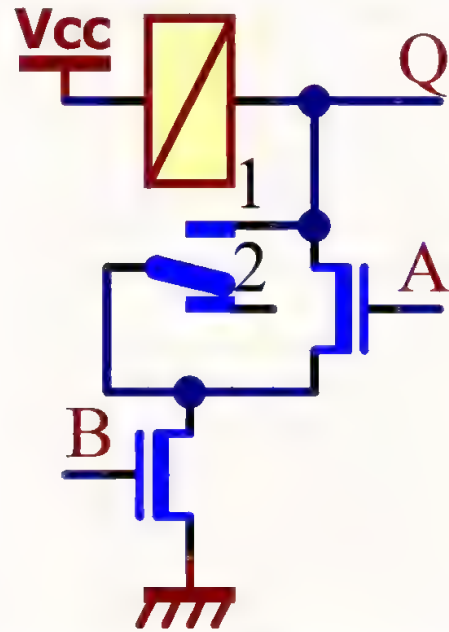
- 40.** Considerând că bobina unui releu permite trecerea valorilor logice prin ea, dar fără a produce scurt-circuit între contactele sale, să se analizeze cum se modifică starea ieșirii logice y la închiderea întrerupătorului K pe schema din dreapta. Ce se va întâmpla după ce întrerupătorul a fost deschis din nou? Să se deseneze diagrama de timp a ieșirii, în funcție de succesiunea stărilor comutatorului. Ce caracte



- 41.** Se conectează un releu în serie cu un bec și o sursă de alimentare, conform schemei alăturate. Deoarece contactele comandate de releu modifică modul de alimentare al releului, se obține o structură secvențială cu reacție. Să se analizeze evoluția schemei în timp și starea becului Q, în funcție de modul cum se modifică pozițiile întrerupătoarelor A și B, respectând ordinea: **AB = 11, 01, 00, 01, 10**. Să se identifice semnificația și nivelul logic activ al intrărilor A secvențial care funcționează la fel.

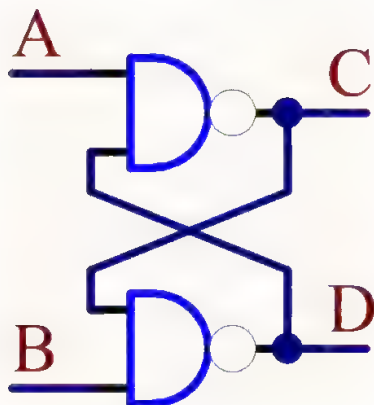


42. Considerând că valorile logice pot trece prin bobina unui relee când celălalt capăt este deconectat, fără a se produce scurt-circuit când la capetele bobinei sunt prezente valori logice diferite, să se explice funcționarea schemei cu reacție din dreapta. Ordinea în care se modifică valorile logice ale semnalelor de intrare aplicate pe grilele tranzistoarelor este: $AB = 11, 01, 00, 01, 10$. În funcție de efectul fiecărui semnal de intrare A și B asupra ieșirii Q, să se identifice rolul și nivelul logic pe care acesta este activ, apoi să se găsească o structură secvențială echivalentă schemei.

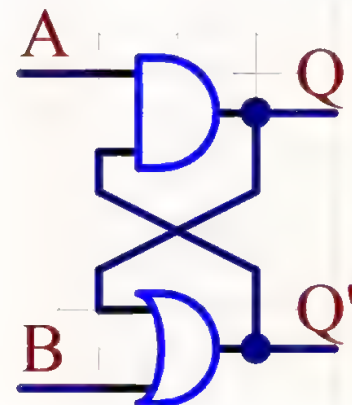


43. Să se analizeze funcționarea celulelor secvențiale pe bază de porți logice din figurile următoare, cu desenarea diagramei de timp și identificarea rolului și nivelului activ (direct/negat) al intrărilor și ieșirilor. Se recomandă aplicarea semnalelor de intrare astfel încât stările porților să se modifice astfel: o poartă blocată și una deschisă, ambele porți deschise, cealaltă blocată și prima deschisă, iar ambele deschise și în final ambele blocate:

a)

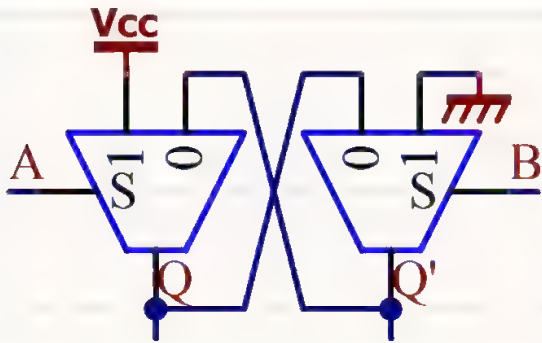


b)

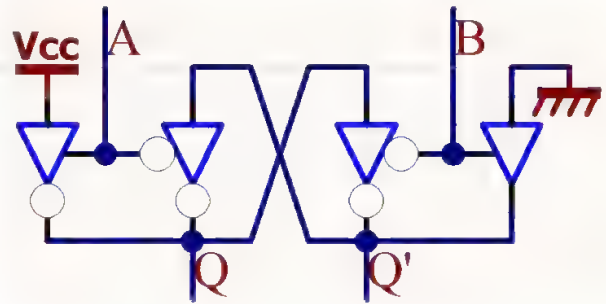


44. Să se analizeze prin diagrame de timp și explicații detaliate în cuvinte funcționarea structurilor logice cu reacție care urmează, identificând semnificația și nivelul activ (direct/negat) pentru fiecare semnal de intrare/ieșire, apoi blocul standard echivalent cu același rol, dacă el există:

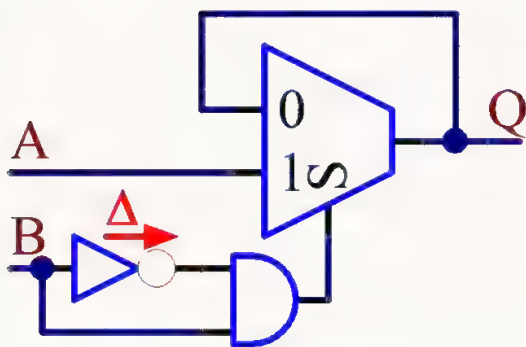
a)



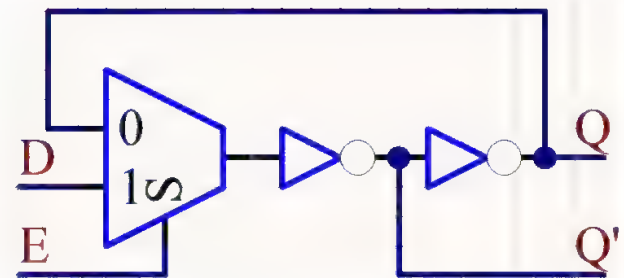
b)



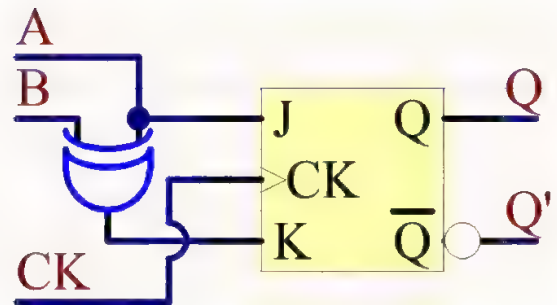
c)



d)

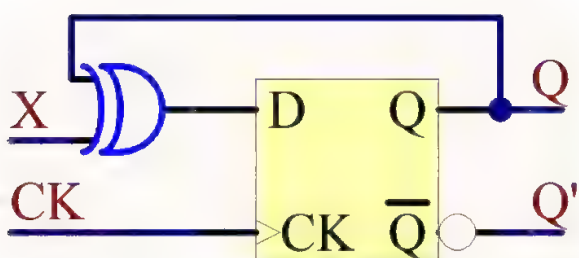


45. Să se analizeze prin explicații detaliate în cuvinte modul de funcționare al structurii secvențiale din schema alăturată, identificând componentele, rolul intrărilor A , B , CK și cu ce anume se poate înlocui ansamblul, astfel încât să se păstreze comportamentul său.

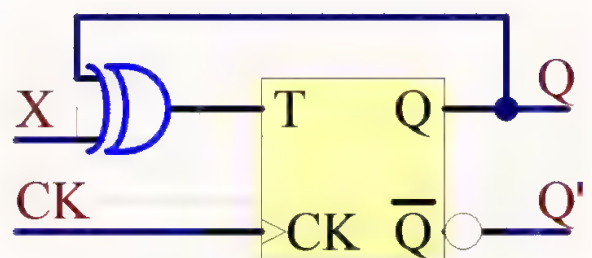


46. Să se analizeze funcționarea schemelor de structuri secvențiale următoare, identificând componentele, apoi denumirile și rolul semnalelor logice X și Q' în cadrul noului ansamblu echivalent obținut:

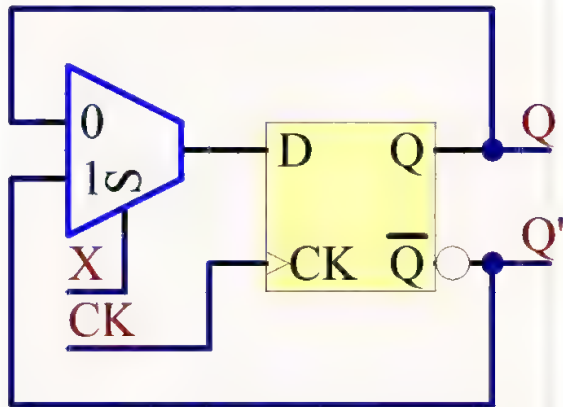
a)



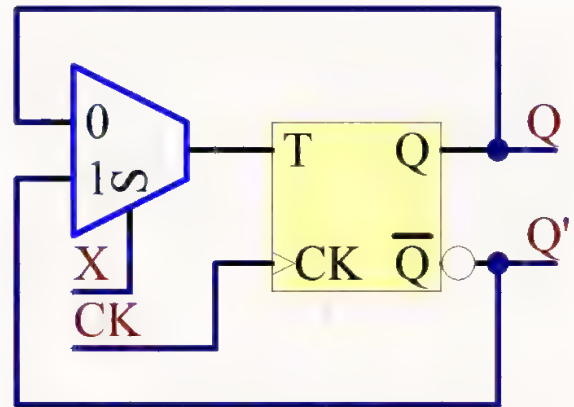
b)



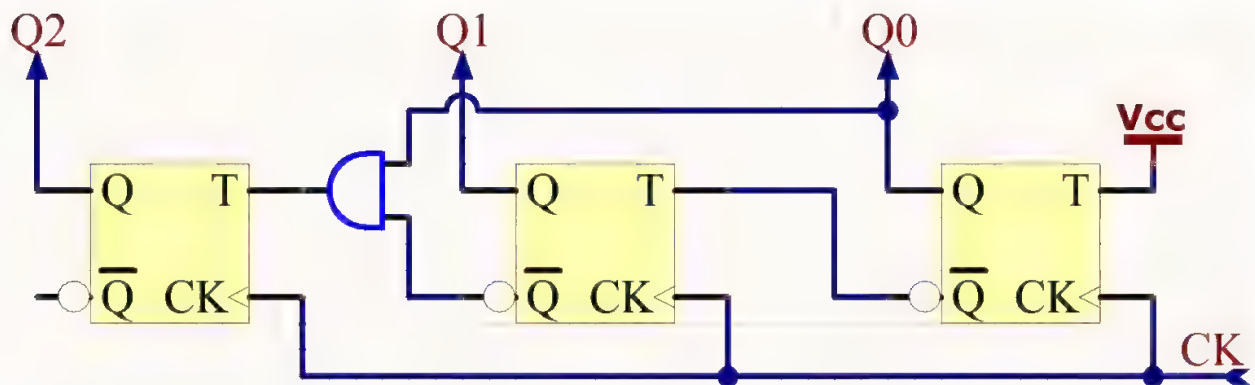
c)



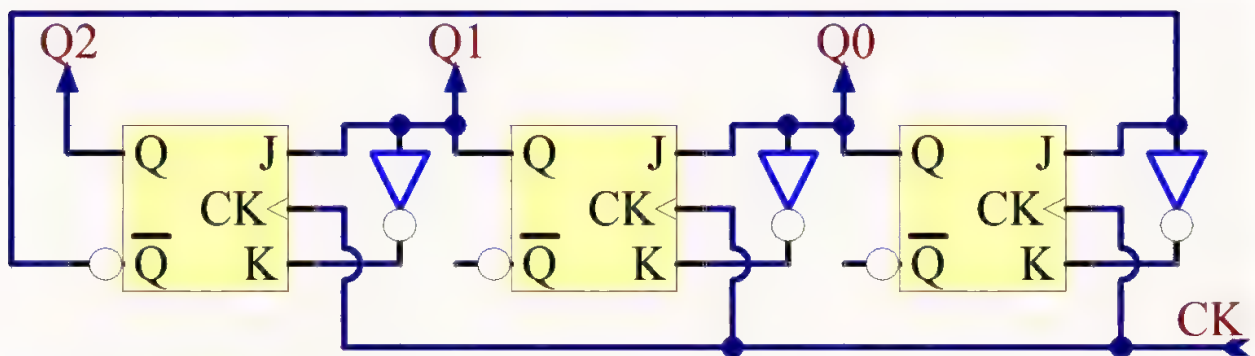
d)



47. Un numărător pe 3 biți este construit conform schemei logice de mai jos. Să se identifice tipul numărătorului, apoi să se explice modul său de funcționare, desenând diagrama de timp și identificând secvența de numărare în zecimal.



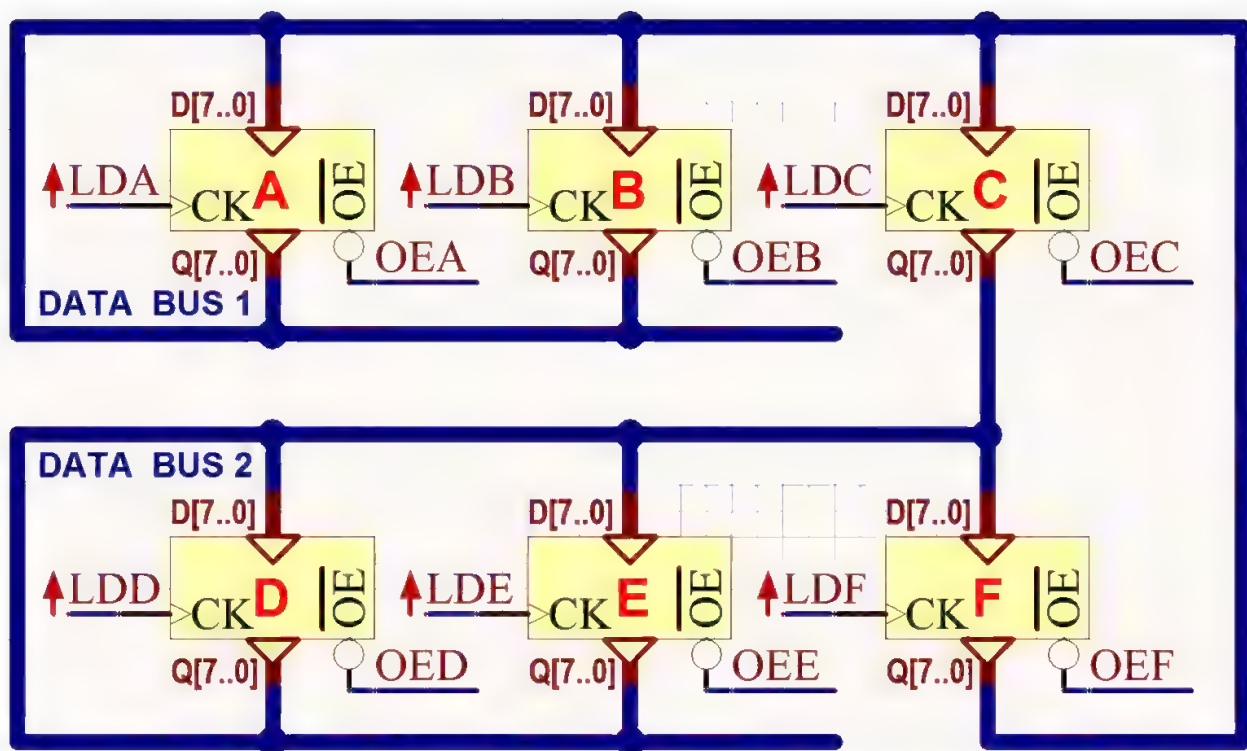
48. Un numărător Johnson de 3 biți este format dintr-un registru de deplasare cu 3 celule și reacție inversată de la ultima ieșire negată la intrare, ca în figura următoare. Considerând că toate celulele au fost inițial resetate, să se analizeze funcționarea ansamblului, prin explicații și diagrame de timp cu toate stările sistemului.



51. Secvențiatorul unui microprocesor de 8 biți furnizează semnalele de comandă “*Load*” și “*Output Enable*” pentru grupul de registre interne cu *încărcare asincronă* conectate la două magistrale interne DATA BUS 1 și DATA BUS 2, conform schemei-bloc din figură. Considerând că la inițializare toate semnalele au fost aduse în starea logică “1”, să se scrie și să se explice secvența semnalelor de comandă necesare pentru a implementa operațiile următoare cu datele din registre, verificând mai întâi dacă respectiva operație este posibilă (*direct, sau folosind și alte registre intermediare*):

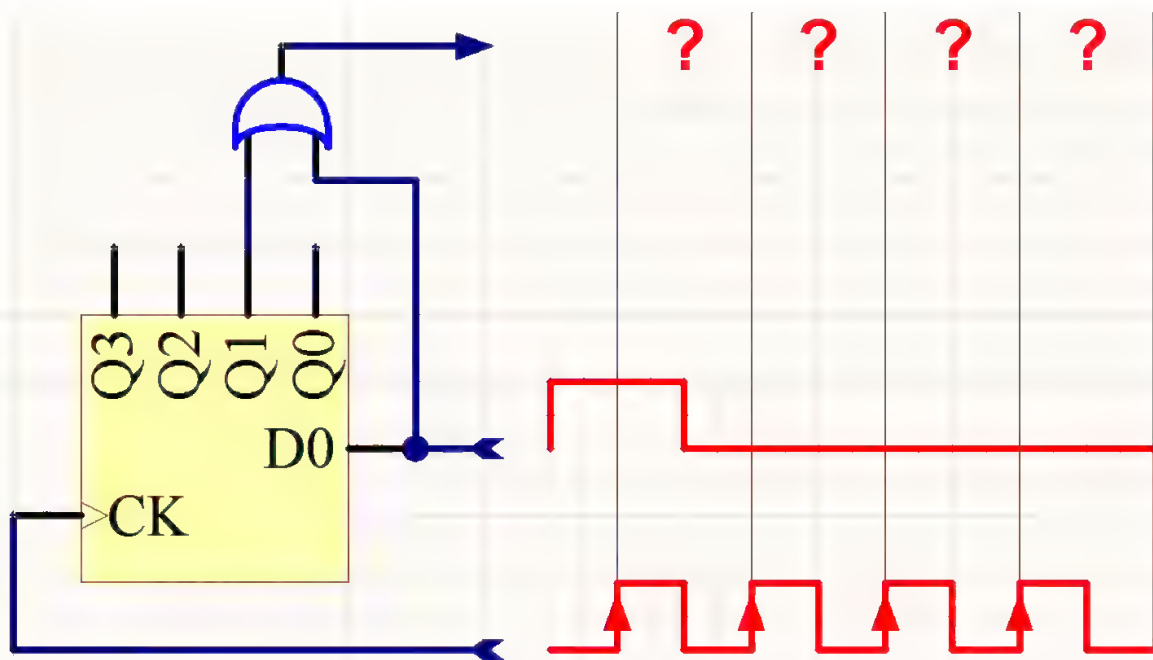
a) **MOV D,A ;**

b) **MOV B,A** simultan cu **MOV E,D** .

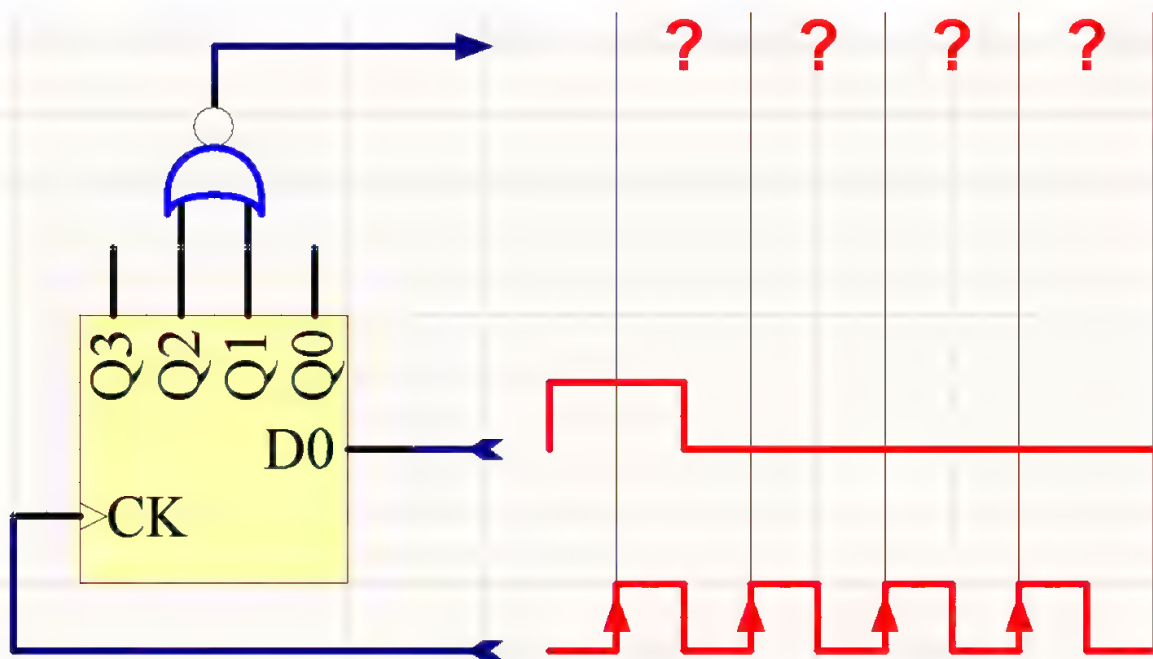


52. La intrarea unui *registru de deplasare spre stânga* de 4 biți, care funcționează sincron pe frontul crescător al ceasului, se aplică un semnal de tip impuls pe durata primei perioade de ceas, iar la ieșiri se conectează câte o poartă logică, ca în figurile următoare. Considerând că registrul de deplasare a fost inițial resetat la valoarea **0000**, să se completeze diagrama de timp cu semnalul corespunzător ieșirii din poarta logică conectată la registrul, pe întreaga durată a semnalelor de intrare:

a)

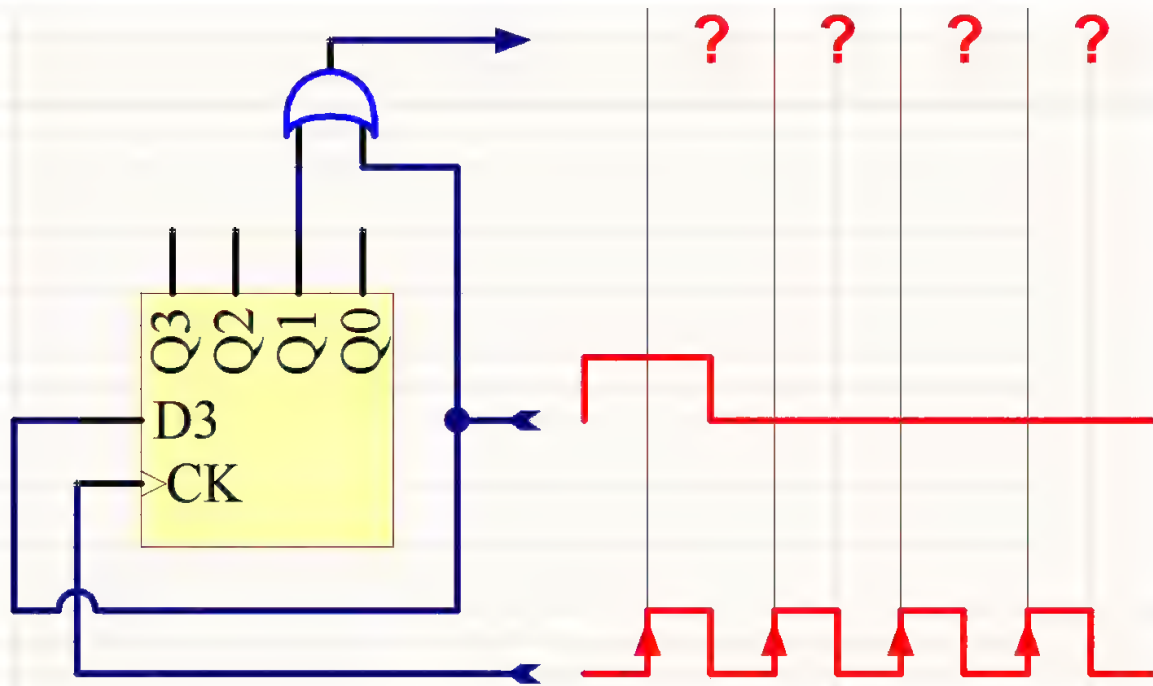


b)

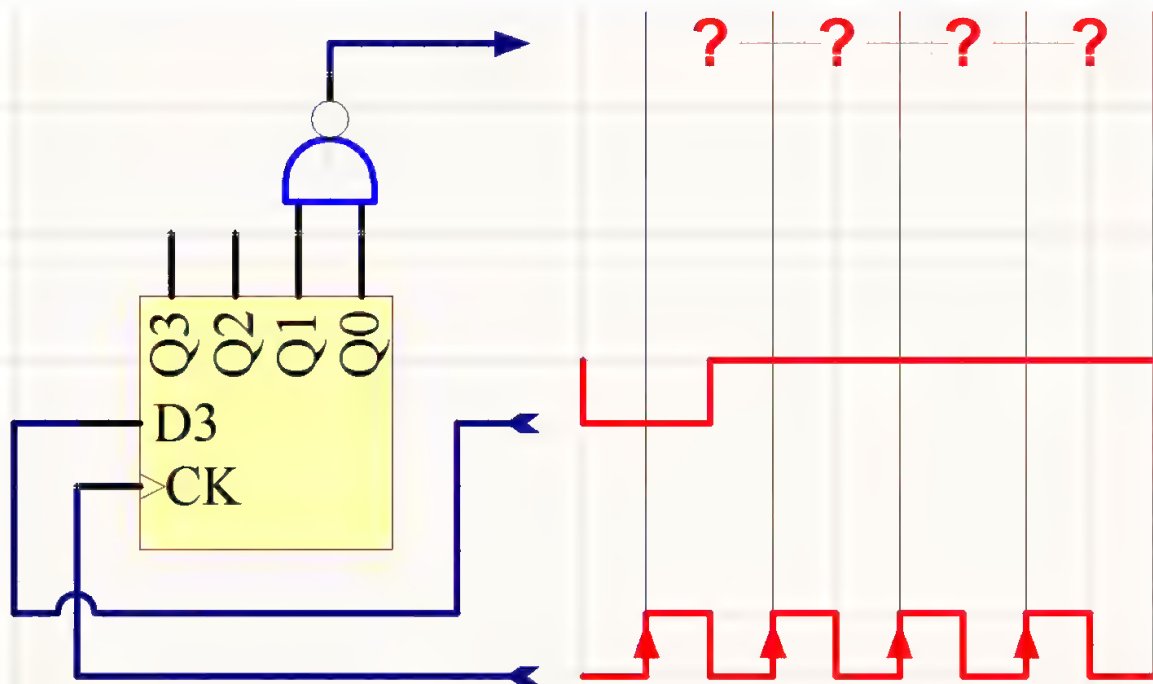


- 53.** La intrarea unui *registru de deplasare spre dreapta* de 4 biți, care funcționează sincron pe frontul crescător al ceasului, se aplică un semnal pe durata mai multor perioade de ceas, iar la ieșiri se conectează câte o poartă logică, ca în figurile următoare. Considerând că registrul conține inițial valoarea 0000, să se completeze diagrama de timp cu semnalul corespunzător ieșirii din poarta logică conectată la registru, pe întreaga durată a semnalelor de intrare:

a)



b)



54. Un microprocesor de tip 8085 execută secvența de instrucțiuni alăturată:

- Să se găsească valorile registrului PC înainte și după execuția instrucțiunilor CALL, respectiv RET;
- Să se găsească adresa unde se află variabila VAR și valorile sale înainte și după execuția programului.

80C0:	LDA	VAR
	CALL	0FE00h
	STA	VAR
	HLT	

FE00:	RLC	
	RET	
VAR:	DB	63h

55. Microprocesorul 8085 execută următorul mic program aflat în memorie:

- a) Să se găsească valorile registrului PC înainte și după execuția instrucțiunilor JMP, respectiv RET;
- b) Să se găsească adresa unde se află variabila VAR și valorile acesteia înainte și după execuția programului.

```

FOE0: LDA  VAR
        JMP  0C800h
        STA  VAR
        HLT
        - - - - -
C800: RRC
        RET
VAR:  DB   0C6h

```

56. Se presupune că înainte de intrarea în programul anterior (problema 55), la adresa de memorie dată de conținutul registrului SP se afla valoarea *E6*, iar la adresa imediat superioară valoarea *F0*. Să se găsească valoarea registrului PC după instrucțiunea RET și valoarea finală a variabilei VAR în această situație.

57. Microprocesorul 8085 execută următorul scurt program aflat în memorie:

- a) Să se arate exact câți octeți sunt necesari în memorie pentru fiecare din instrucțiunile programului;
- b) Să se găsească valorile hexa-zecimale ale registrului PC înainte și după execuția instrucțiunilor CALL, respectiv RET (de calculat 4 numere);
- c) Să se găsească adresa unde se află variabila VAR și valorile acesteia înainte și după execuția programului (de calculat 3 numere).

```

A0B0: LDA  VAR
        CALL 0CCCCCh
        STA  VAR
        HLT
        - - - - -
CCCC: RLC
        RET
VAR:  DB   11h

```

58. Microprocesorul 8085 execută următoarea secvență de instrucțiuni dintr-un program aflat în memorie:

- a) Să se arate exact câți octeți sunt stocați în memorie pentru fiecare din instrucțiunile programului;
- b) Să se găsească valorile hexa-zecimale ale registrului PC înainte și după execuția instrucțiunilor CALL, respectiv RET (de calculat 4 numere);
- c) Să se găsească adresa de memorie a variabilei VAR și conținutul variabilei înainte și după execuția programului (de calculat 3 numere).

```

8ABC: LDA  VAR
        CALL 0DBDBh
        STA  VAR
        HLT
        - - - - -
DBDB: RRC
        RET
VAR:  DB   88h

```

59. Microprocesorul 8085 execută următorul mic program alăturat, aflat în memorie:

Considerând numărul locațiilor de memorie ocupate de instrucțiunile prezentate ale programului, să se găsească și să se explice valorile registrului PC înainte și după execuția celor două instrucțiuni JMP (de determinat 4 numere).

```

80C0: JMP 0ABCDh
BACK: MOV B,A
      -----
ABCD: MVI A,1Eh
      JMP BACK

```

60. Microprocesorul 8085 execută programul alăturat din memorie:

Considerând numărul locațiilor de memorie ocupate de fiecare din instrucțiunile programului, să se găsească și să se explice care sunt valorile registrului PC înainte de extragerea din memorie a codurilor instrucțiunilor CALL și RET, precum și după execuția acestora (de determinat 4 numere).

```

F0E0: CALL 0C800h
      STA 1234h
      -----
C800: RLC
      RET

```

61. Pentru fiecare dintre cele două programe prezentate în continuare, executate de microprocesorul 8085, să se explice cum se determină următoarele:

- Numărul de octeți care sunt folosiți pentru a codifica în memorie fiecare instrucțiune;
- Cele 4 valori care reprezintă conținutul registrului PC înainte și după execuția instrucțiunilor CALL, respectiv RET;
- Cele 3 numere care reprezintă adresa locației de memorie unde se află variabila VAR și conținutul acestei locații înainte și după execuția fiecărui program.

```

1111: LDA VAR
      MOV B,A
      CALL 8085h
      STA VAR
      HLT
      -----
8085: MOV A,B
      RRC
      RET
VAR: DB 11h

```

```

3333: LDA VAR
      ORA B
      CALL 8051h
      STA VAR
      HLT
      -----
8051: MVI A,11h
      RLC
      RET
VAR: DB 0BDh

```

62. Să se trateze *reprezentarea numerelor în virgulă mobilă* pe cazul general, cu toate aspectele importante, fără a particulariza pentru un anumit număr de biți.

63. Folosind explicații și desene, să se trateze *operațiile logice cu măști* pe mai mulți biți care se pot utiliza la programarea sistemelor de calcul.
64. Folosind explicații și desene, să se trateze *operațiile logice de rotație* care se pot utiliza la programarea sistemelor de calcul.
65. Să se deseneze simbolurile pentru tranzistoarele utilizate drept comutatoare comandate, apoi să se explice care valori logice pot trece prin ele, pe unde trec și în ce condiții.
66. Să se trateze construcția și modul de funcționare a *etajului de ieșire* pentru porțile logice.
67. Folosind explicații și scheme, să se trateze alcătuirea internă la nivelul etajului de ieșire și modul de funcționare al *inversoarelor cu trei stări ce se deconectează pe zero*.
68. Folosind explicații și scheme, să se trateze alcătuirea internă la nivelul etajului de ieșire și modul de funcționare al *inversoarelor cu trei stări ce se deconectează pe unu*.
69. Folosind explicații și desene, să se trateze *modul de funcționare și alcătuirea internă a decodificatoarelor de adresă* utilizate la construcția sistemelor de calcul.
70. Folosind explicații și desene, să se trateze *modul de funcționare și alcătuirea internă a multiplexoarelor* utilizate la construcția sistemelor de calcul.
71. Folosind explicații și desene, să se trateze *modul de funcționare și construcția demultiplexoarelor* logice.
72. Să se explice detaliat cum se poate perfecționa celula de memorare RS cu porți logice, prin îngustarea intervalului de timp în care intrările pot influența starea celulei (obținerea “*latch*”-ului RS transparent pe palier și a *bistabilului* RS sincron pe front)
73. Să se deseneze schema și diagramele de timp cu evidențierea întârzierilor de propagare pentru un *numărător asincron descrescător* cu 3 ranguri binare.

74. Să se deseneze schema și diagramele de timp cu evidențierea întârzierilor de propagare pentru un *numărător sincron descrescător* cu 3 ranguri binare.
75. Să se explice, desenând simbolul echivalent, tabelul de adevăr și diagramele de timp, cum funcționează o memorie SRAM cu organizarea **32K x 8biți**, la citire și scriere.
76. Să se trateze pe scurt organizarea generală a sistemelor de calcul: schema-bloc cu componentele principale și semnalele de interconectare, precizând rolul și modul lor de funcționare.
77. Să se prezinte arhitectura simplificată (fără interconexiunile interne și externe) a microprocesorului 8085, explicând detaliat denumirea, dimensiunea în biți și funcțiile *unității de calcul, a indicatorilor de condiții, a registrelor generale și a celor cu funcții speciale*.
78. Să se explice, desenând schema și diagramele de timp, cum se pot separa *adresele* și *datele* de pe magistrala inferioară combinată AD[7..0] a microprocesorului 8085.
79. Să se comenteze modul de funcționare al microprocesorului 8085, prin analiza pașilor în care este extrasă din memorie și executată instrucțiunea **LDA VAR**, dacă variabila *VAR* este plasată la adresa **4C5Bh**.
80. Să se explice cum se modifică *indicatoarele de condiții Cy și Z* ale unui microprocesor în urma execuției instrucțiunilor *aritmetice* și instrucțiunilor *logice*.
81. Să se explice cum se modifică *registrele PC, SP și stiva* unui microprocesor în momentul execuției instrucțiunii “*apel de subrutină*”.
82. Să se explice pe larg, prin desene exemplificative comentate, funcționarea stivei microprocesorului 8085 la *păstrarea temporară a datelor*.
83. Să se explice pe larg, prin desene exemplificative comentate, funcționarea stivei microprocesorului 8085 la *lucrul cu subrutine și întreruperi*.

Bibliografie selectivă pe capitole

Baruch Z. F. – “ <i>Structura sistemelor de calcul</i> ”, Editura Albastră, Cluj-Napoca, 2004.	cap. 4, 13, 14
Cleju I. – “ <i>Circuite logice. Proiectare și aplicații</i> ”, Editura Gh. Asachi, Iași, 2002.	cap. 8 - 12
Hayes J. P. – “ <i>Computer Architecture and Organisation</i> ”, 3 rd edition, McGraw-Hill, 2002.	cap. 4, 13, 14
Harris D. M. , Harris S. L. – “ <i>Digital Design and Computer Architecture</i> ”, 2 nd edition, Elsevier, 2013.	cap. 1 - 14, 19
Huțanu C. , Postolache M. – “ <i>Sisteme cu microprocesoare în conducerea automată a proceselor</i> ”, vol. I, ediția a II-a, Editura Academica, Iași, 2001.	cap. 13 - 18
Kumar K. U. , Umashankar B. S. – “ <i>The 8085 Microprocessor: Architecture, Programming and Interfacing</i> ”, Pearson, 2008.	cap. 1 - 19
Lupu C. , Stăncescu S. – “ <i>Microprocesoare. Circuite. Proiectare.</i> ”, Editura Militară, București, 1986.	cap. 14 - 18
Mano M. M. , Kime C. R. – “ <i>Logic and Computer Design Fundamentals</i> ”, 3 rd edition, Prentice Hall, 2004.	cap. 1 - 15
Manta V. , Ungureanu F. – “ <i>Introducere în știința sistemelor și a calculatoarelor</i> ”, vol. I, Editura Gh. Asachi, Iași, 2002.	cap. 1 - 7
Mureșan T. , Strugaru C. , Stoinescu R. , Petriu E. – “ <i>Microprocesorul 8080 în aplicații</i> ”, Editura Facla, Timișoara, 1981.	cap. 14 - 18
Patrubány M. – “ <i>Totul despre microprocesorul Z80</i> ”, vol. I și vol. II, Editura Tehnică, București, 1989.	cap. 14 - 18
Patterson D. A. , Hennessy J. L. – “ <i>Computer Organisation and Design. The Hardware / Software Interface</i> ”, 4 th edition, Elsevier, 2009.	cap. 3, 4, 13, 14

Pleter O. , Constantinescu C. – “ <i>Z80 în microcalculatoare compatibile Spectrum</i> ”, Editura Militară, București, 1995.	cap. 13 - 18
Potorac A. D. – “ <i>Bazele proiectării circuitelor numerice</i> ”, Editura Matrix-Rom, București, 2002.	cap. 3 - 12
Srinath N. K. – “ <i>8085 Microprocessor Programming and Interfacing</i> ”, Prentice-Hall, 2005.	cap. 13 - 18
Sztojanov I. , Borcoci E. , Tomescu N. , Bulik D. , Petrec M. , Petrec C. – “ <i>De la poarta TTL la microprocesor</i> ”, vol. I și vol. II. , Editura Tehnică, București, 1987.	cap. 5 - 18
Toacșe Gh. – “ <i>Introducere în microprocesoare</i> ”, Editura Științifică și Enciclopedică, București, 1986.	cap. 1 - 18
Toacșe Gh. , Nicula D. – “ <i>Electronică digitală</i> ”, Editura Teora, București, 1996.	cap. 5 - 12
Toacșe Gh. , Nicula D. – “ <i>Electronică digitală</i> ”, vol. I, Editura Tehnică, București, 2005.	cap. 5 - 13
Valachi Al. , Hoza F. , Onofrei V. , Silion R. – “ <i>Analiza, sinteza și testarea dispozitivelor numerice</i> ”, Editura Nord-Est, Iași, 1993.	cap. 3 - 12
Weste N. H. E. , Harris D. M. – “ <i>CMOS VLSI Design. A Circuits and Systems Perspective</i> ”, 4 th edition, Pearson, 2011.	cap. 8, 19
* * * – “ <i>Component Data and Catalog</i> ”, Intel Corporation Ltd. , Santa Clara, 1977, 1980, 1982.	cap. 15, 16, 17
* * * – “ <i>High Speed CMOS Logic Data Book</i> ”, Texas Instruments Inc. , Dallas, 1993.	cap. 18
* * * – “ <i>Universal Trainer Reference Manual</i> ”, rev. 1.3, EMAC Inc. , Carbondale, 1993.	cap. 15 - 18

“ Lucrarea constituie o prezentare a unor cunoștințe fundamentale, atât teoretice cât și practice, necesare specialiștilor din diverse ramuri ale ingineriei. Sunt prezentate aspecte teoretice la un nivel accesibil, cu accent preponderent pe înțelegerea intuitivă a fenomenelor, împreună cu o serie de exemple. Lucrarea pune la dispoziție un material unitar, destinat însușirii unor elemente fundamentale privitoare la funcționarea sistemelor de calcul. Materialul oferit este autonom, permițând parcurgerea și înțelegerea noțiunilor fără a necesita consultarea unor lucrări adiționale.”

Prof. univ. dr. ing. Alexandru Onea

